PCI2006 数据采集卡

WIN2000/XP 驱动程序使用说明书



北京阿尔泰科技发展有限公司
产品研发部修订

请您务必阅读《使用纲要》，他会使您事半功倍!

目 录

第一章 版权信息与命名约定

第一节、版权信息

本软件产品及相关套件均属北京阿尔泰科技发展有限公司所有，其产权受国家法律绝对保护，除非本公司书面允许，其他公司、单位、我公司授权的代理商及个人不得非法使用和拷贝，否则将受到国家法律的严厉制裁。您若需要我公司产品及相关信息请及时与当地代理商联系或直接与我们联系，我们将热情接待。

第二节、命名约定

一、为简化文字内容，突出重点，本文中提到的函数名通常为基本功能名部分，其前缀设备名如 PCIxxxx_则被省略。如 PCI2006_CreateDevice 则写为 CreateDevice。

二、函数名及参数中各种关键字缩写

缩写	全称	汉语意思	缩写	全称	汉语意思
Dev	Device	设备	DI	Digital Input	数字量输入
Pro	Program	程序	DO	Digital Output	数字量输出
Int	Interrupt	中断	CNT	Counter	计数器
Dma	Direct Memory Access	直接内存存取	DA	Digital convert to Analog	数模转换
AD	Analog convert to Digital	模数转换	DI	Differential	(双端或差分) 注：在常量选项中
Npt	Not Empty	非空	SE	Single end	单端
Para	Parameter	参数	DIR	Direction	方向
SRC	Source	源	ATR	Analog Trigger	模拟量触发
TRIG	Trigger	触发	DTR	Digital Trigger	数字量触发
CLK	Clock	时钟	Cur	Current	当前的
GND	Ground	地	OPT	Operate	操作
Lgc	Logical	逻辑的	ID	Identifier	标识
Phys	Physical	物理的			

第二章 使用纲要

第一节、使用上层用户函数，高效、简单

如果您只关心通道及频率等基本参数，而不必了解复杂的硬件知识和控制细节，那么我们强烈建议您使用上层用户函数，它们就是几个简单的形如Win32 API的函数，具有相当的灵活性、可靠性和高效性。诸如InitDeviceProAD、ReadDeviceProAD_NotEmpty、SetDeviceDO等。而底层用户函数如WriteRegisterULong、ReadRegisterULong、WritePortByte、ReadPortByte……则是满足了解硬件知识和控制细节、且又需要特殊复杂控制的用户。但不管怎样，我们强烈建议您使用上层函数（在这些函数中，您见不到任何设备地址、寄存器端口、中断号等物理信息，其复杂的控制细节完全封装在上层用户函数中。）对于上层用户函数的使用，您基本上不必参考硬件说明书，除非您需要知道板上D型插座等管脚分配情况。

第二节、如何管理 PCI 设备

由于我们的驱动程序采用面向对象编程，所以要使用设备的一切功能，则必须首先用CreateDevice函数创建一个设备对象句柄hDevice，有了这个句柄，您就拥有了对该设备的绝对控制权。然后将此句柄作为参数传递给相应的驱动函数，如InitDeviceProAD可以使用hDevice句柄以程序查询方式初始化设备的AD部件，ReadDeviceProAD_NotEmpty (或ReadDeviceProAD_Half)函数可以用hDevice句柄实现对AD数据的采样读取，SetDeviceDO函数可用实现开关量的输出等。最后可以通过ReleaseDevice将hDevice释放掉。

第三节、如何用非空查询方式取得 AD 数据

当您有了hDevice设备对象句柄后，便可用InitDeviceProAD函数初始化AD部件，关于采样通道、频率等参数的设置是由这个函数的pADPara参数结构体决定的。您只需要对这个pADPara参数结构体的各个成员简单赋值即可实现所有硬件参数和设备状态的初始化。然后用StartDeviceProAD即可启动AD部件，开始AD采样，然后便可用ReadDeviceProAD_NotEmpty反复读取AD数据以实现连续不间断采样。当您需要暂停设备时，执行StopDeviceProAD，当您需要关闭AD设备时，ReleaseDeviceProAD便可帮您实现（但设备对象hDevice依然存在）。

（注：ReadDeviceProAD_ NotEmpty虽然主要面对批量读取、高速连续采集而设计，但亦可用它以单点或几点的方式读取AD数据，以满足慢速、高实时性采集需要）。具体执行流程请看下面的图 2.1.1。

第四节、如何用半满查询方式取得 AD 数据

当您有了hDevice设备对象句柄后，便可用InitDeviceProAD函数初始化AD部件，关于采样通道、频率等参数的设置是由这个函数的pADPara参数结构体决定的。您只需要对这个pADPara参数结构体的各个成员简单赋值即可实现所有硬件参数和设备状态的初始化。然后用StartDeviceProAD即可启动AD部件，开始AD采样，接着调用GetDevStatusProAD函数以查询AD的存储器FIFO的半满状态，如果达到半满状态，即可用ReadDeviceProAD_Half函数读取一批半满长度（或半满以下）的AD数据，然后接着再查询FIFO的半满状态，若有效再读取，就这样反复查询状态反复读取AD数据即可实现连续不间断采样。当您需要暂停设备时，执行StopDeviceProAD，当您需要关闭AD设备时，ReleaseDeviceProAD便可帮您实现（但设备对象hDevice依然存在）。（注：ReadDeviceProAD_Half函数在半满状态有效时也可以单点或几点的方式读取AD数据，只是到下一次半满信号到来时的时间间隔会变得非常短，而不再是半满间隔。）具体执行流程请看下面的图 2.1.2。

第五节、如何用中断方式取得 AD 数据

当您有了hDevice设备对象句柄后，便可用InitDeviceIntAD函数初始化AD部件，关于采样通道、频率等的参数的设置是由这个函数的pPara参数结构体决定的。您只需要对这个pPara参数结构体的各个成员简单赋值即可实现所有硬件参数和设备状态的初始化。同时应调用CreateSystemEvent函数创建一个内核事件对象句柄hEvent赋给InitDeviceIntAD的相应参数，它将作为接受AD半满中断事件的变量。然后用StartDeviceIntAD即可启动AD部件，开始AD采样，接着调用Win32 API函数WaitForSingleObject等待hEvent中断事件的发生，在中断未到时，自动使所在线程进入睡眠状态（不消耗CPU时间），反之，则立即唤醒所在线程，执行它下面的代码，此时您便可用ReadDeviceIntAD函数一批半满长度（或半满以下）的AD数据，然后再接着再等待FIFO的半满中断事件，若有效再读取，就这样反复读取AD数据即可实现连续不间断采样。当您需要暂停设备时，执行StopDeviceIntAD，当您需要关闭AD设备时，ReleaseDeviceIntAD便可帮您实现（但设备对象hDevice依然存在）。（注：ReadDeviceIntAD函数在半满中断事件发生时可以单点或几点的方式读取AD数据，只是到下一次半满中断事件到来时的时间间隔会变得非常短，而不再是半满间隔，但它不同于半满查询方式读取，由于半满中断属于硬件中断，其优先级别高于所有软件，所以您单点或几点读取AD数据时，千万不能让中断间隔太短，否则，有可能使您的整个系统被半满中断事件吞没，就象死机一样，不能动弹。 切忌、切忌！）具体执行流程请看图 2.1.3。

注意：图中较粗的虚线表示对称关系。如红色虚线表示CreateDevice和ReleaseDevice两个函数的关系是：最初执行一次CreateDevice，在结束是就须执行一次ReleaseDevice。

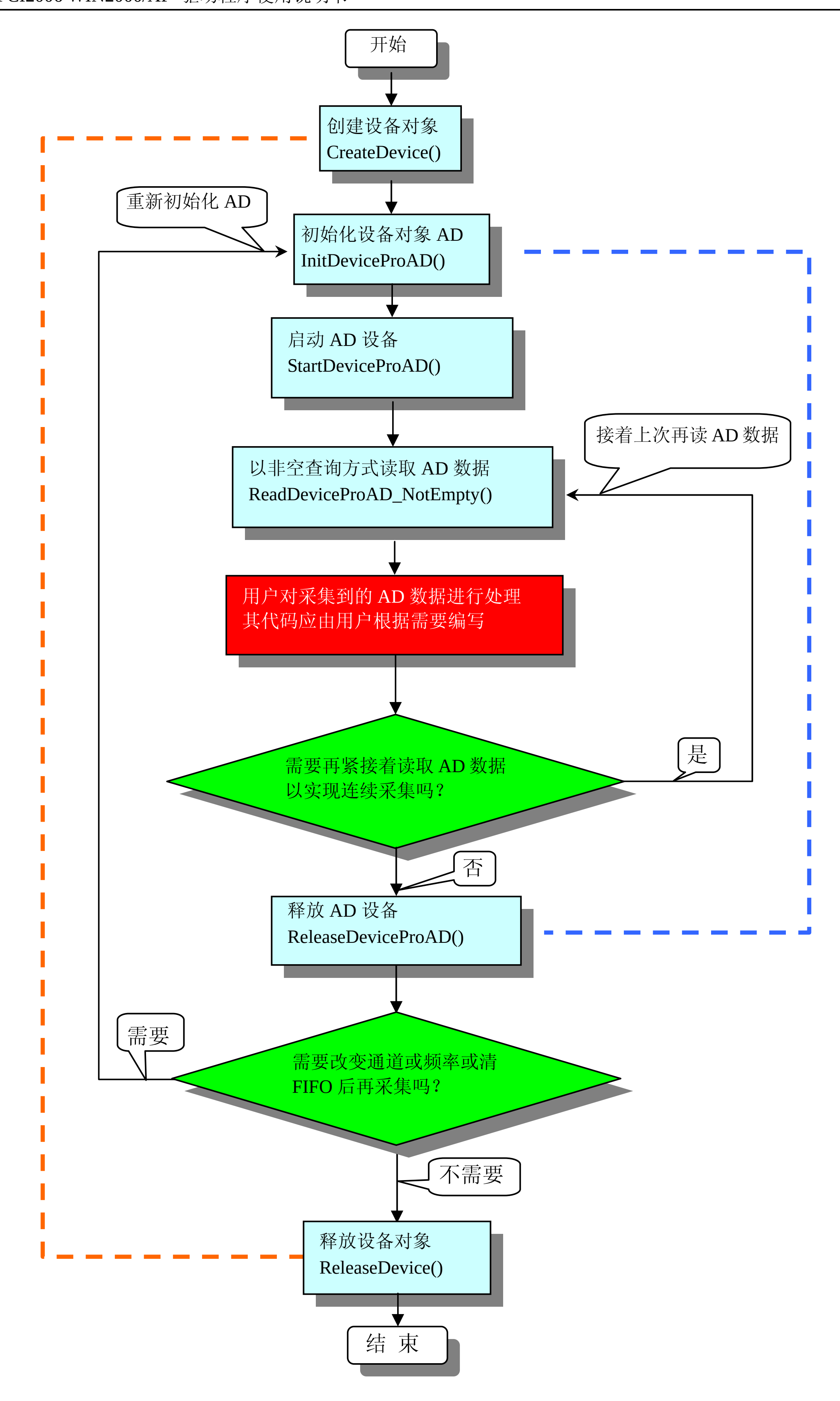


图 2.1.1 非空查询方式 AD 采集过程

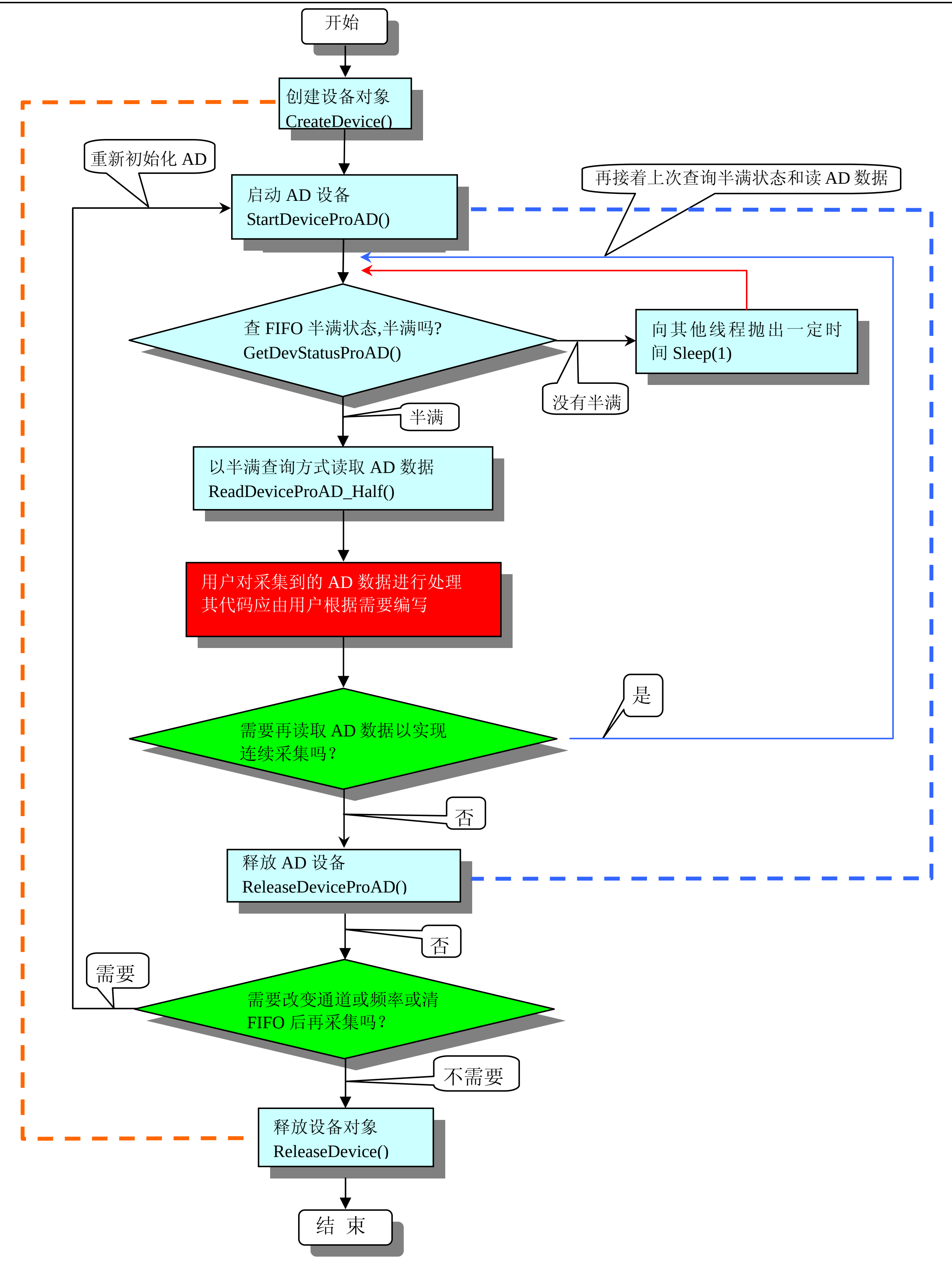


图 2.1.2 半满查询方式 AD 采集过程

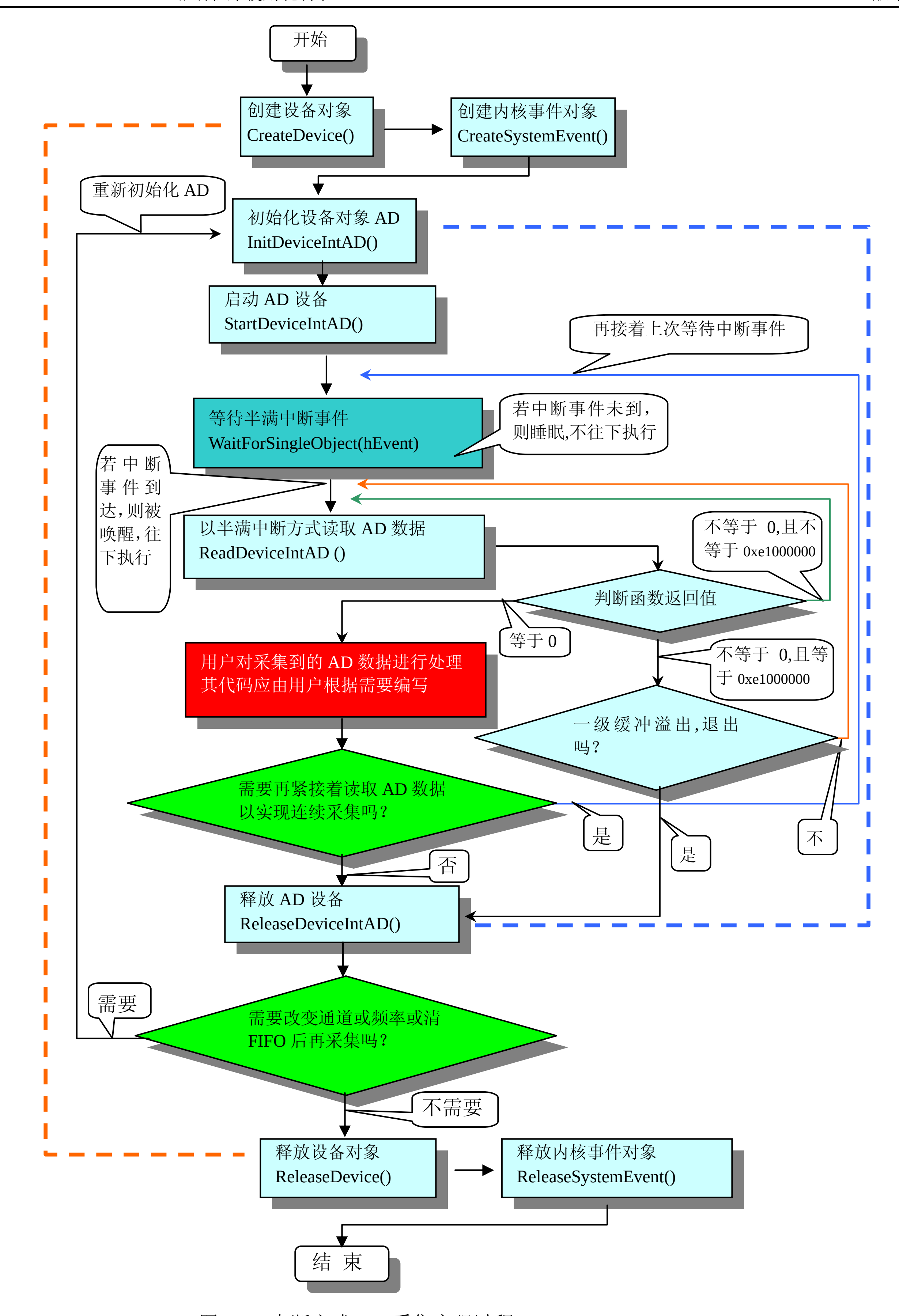


图 2.1.3 中断方式 AD 采集实现过程

第六节、如何实现开关量的简便操作

当您有了hDevice设备对象句柄后，便可用SetDeviceDO函数实现开关量的输出操作，其各路开关量的输出状态由其bDOSts[16]中的相应元素决定。由GetDeviceDI函数实现开关量的输入操作，其各路开关量的输入状态由其bDISts[16]中的相应元素决定。

第七节、哪些函数对您不是必须的

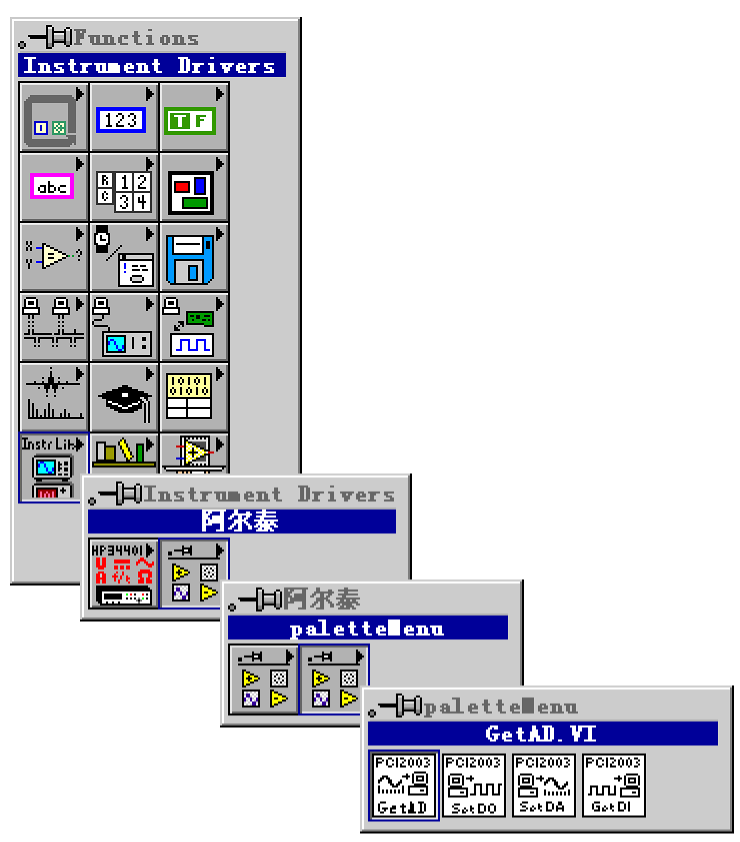
公共函数如CreateFileObject， WriteFile，ReadFile等一般来说都是辅助性函数，除非您要使用存盘功能。如果您使用上层用户函数访问设备，那么GetDeviceAddr， WriteRegisterByte， WriteRegisterWord，WriteRegisterULong， ReadRegisterByte， ReadRegisterWord， ReadRegisterULong等函数您可完全不必理会，除非您是作为底层用户管理设备。而WritePortByte， WritePortWord， WritePortULong， ReadPortByte，ReadPortWord， ReadPortULong则对PCI用户来讲，可以说完全是辅助性，它们只是对我公司驱动程序的一种功能补充，对用户额外提供的，它们可以帮助您在NT、Win2000 等操作系统中实现对您原有传统设备如ISA卡、串口卡、并口卡的访问，而没有这些函数，您可能在基于Windows NT架构的操作系统中无法继续使用您原有的老设备。

第三章 PCI 即插即用设备操作函数接口介绍

由于我公司的设备应用于各种不同的领域，有些用户可能根本不关心硬件设备的控制细节，只关心AD的首末通道、采样频率等，然后就能通过一两个简易的采集函数便能轻松得到所需要的AD数据。这方面的用户我们称之为上层用户。那么还有一部分用户不仅对硬件控制熟悉，而且由于应用对象的特殊要求，则要直接控制设备的每一个端口，这是一种复杂的工作，但又是必须的工作，我们则把这一群用户称之为底层用户。因此总的看来，上层用户要求简单、快捷，他们最希望在软件操作上所要面对的全是他们最关心的问题，比如在正式采集数据之前，只须用户调用一个简易的初始化函数（如InitDeviceProAD）告诉设备我要使用多少个通道，采样频率是多少赫兹等，然后便可以用ReadDeviceProAD_ NotEmpty（或ReadDeviceProAD_Half）函数指定每次采集的点数，即可实现数据连续不间断采样。而关于设备的物理地址、端口分配及功能定义等复杂的硬件信息则与上层用户无任何关系。那么对于底层用户则不然。他们不仅要关心设备的物理地址，还要关心虚拟地址、端口寄存器的功能分配，甚至每个端口的Bit位都要了如指掌，看起来这是一项相当复杂、繁琐的工作。但是这些底层用户一旦使用我们提供的技术支持，则不仅可以让您不必熟悉PCI总线复杂的控制协议，同是还可以省掉您许多繁琐的工作，比如您不用去了解PCI的资源配置空间、PNP即插即用管理，而只须用GetDeviceAddr函数便可以同时取得指定设备的物理基地址和虚拟线性基地址。这个时候您便可以用这个虚拟线性基地址，再根据硬件使用说明书中的各端口寄存器的功能说明，然后使用ReadRegisterULong和WriteRegisterULong对这些端口寄存器进行 32 位模式的读写操作，即可实现设备的所有控制。

综上所述，用户使用我公司提供的驱动程序软件包将极大的方便和满足您的各种需求。但为了您更省心，别忘了在您正式阅读下面的函数说明时，先明白自己是上层用户还是底层用户，因为在《设备驱动接口函数总列表》中的备注栏里明确注明了适用对象。

另外需要申明的是，在本章和下一章中列明的关于 LabView 的接口，均属于外挂式驱动接口，他是通过LabView 的 Call Labrary Function 功能模板实现的。它的特点是除了自身的语法略有不同以外，每一个基于LabView 的驱动图标与 Visual C++、Visual Basic、Delphi 等语言中每个驱动函数是一一对应的，其调用流程和功能是完全相同的。那么相对于外挂式驱动接口的另一种方式是内嵌式驱动。这种驱动是完全作为 LabView 编程环境中的紧密耦合的一部分，它可以直接从 LabView 的 Functions 模板中取得，如下图所示。此种方式更适合上层用户的需要，它的最大特点是方便、快捷、简单，而且可以取得它的在线帮助。关于 LabView 的外挂式驱动和内嵌式驱动更详细的叙述，请参考 LabView 的相关演示。



LabView 内嵌式驱动接口的获取方法

第一节、设备驱动接口函数总列表（每个函数省略了前缀“PCI2006_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建 PCI 设备对象(用设备逻辑号)	上层及底层用户
GetDeviceCount	取得同一种 PCI 设备的总台数	上层及底层用户
GetDeviceCurrentID	取得指定设备句柄指向的设备 ID 号	上层及底层用户
ListDeviceDlg	列表所有同一种 PCI 设备的各种配置	上层及底层用户
ReleaseDevice	关闭设备，且释放 PCI 总线设备对象	上层及底层用户
② **AD** 的程序方式读取函数		
InitDeviceProAD	初始化 AD 部件准备传输	上层用户
StartDeviceProAD	启动 AD 设备，开始转换	上层用户
SetDevFreqenceAD	可动态改变 AD 采样频率	上层用户
ReadDeviceProAD_NotEmpty	连续读取当前 PCI 设备上的 AD 数据	上层用户
GetDevStatusProAD	取得当前 PCI 设备 FIFO 半满状态	上层用户
ReadDeviceProAD_Half	连续批量读取 PCI 设备上的 AD 数据	上层用户
StopDeviceProAD	暂停 AD 设备	上层用户
ReleaseDeviceProAD	释放设备上的 AD 部件	上层用户
③ 中断方式 **AD** 读取函数（唯有此种方式采用强制二级队列缓冲和动态链表技术）		
InitDeviceIntAD	初始化 PCI 设备 AD 部件，如通道等	上层用户
StartDeviceIntAD	启动 AD 采集	上层用户
ReadDeviceIntAD	连续批量读取 PCI 设备上的 AD 数据	上层用户
StopDeviceIntAD	停止 AD 采集	上层用户
ReleaseDeviceIntAD	释放设备上的 AD 部件	上层用户
④ **AD** 硬件参数系统保存、读取函数		
LoadParaAD	从 Windows 系统中读入硬件参数	上层用户
SaveParaAD	往 Windows 系统写入设备硬件参数	上层用户
⑤DA 模拟量输出操作函数		

InitDeviceProDA	初始化 DA	
WriteDeviceProDA	输出 DA 数据	
DIO 开关量简易操作函数		
GetDeviceDI	开关输入函数	上层用户
SetDeviceDO	开关输出函数	上层用户

使用需知：

Visual C++ & C++Builder：

要使用如下函数关键的问题是：

首先，必须在您的源程序中包含如下语句：

#include “C:\Art\PCI2006\INCLUDE\PCI2006.H”

注：以上语句采用默认路径和默认板号，应根据您的板号和安装情况确定 PCI2006.H 文件的正确路径，当然也可以把此文件拷到您的源程序目录中。然后加入如下语句：

#include “PCI2006.H”

另外，要在 VB 环境中用子线程以实现高速、连续数据采集与存盘，请务必使用 VB5.0 版本。当然如果您有 VB6.0 的最新版，也可以实现子线程操作。

C++ Builder:

要使用如下函数一个关键的问题是首先必须将我们提供的头文件(PCI2006.H)写进您的源程序头部。如：#include “\Art\PCI2006\Include\PCI2006.h”，然后再将 PCI2006.Lib 库文件分别加入到您的 C++ Builder 工程中。其具体办法是选择 C++ Builder 集成开发环境中的工程(Project)菜单中的“添加”（Add to Project）命令，在弹出的对话框中分别选择文件类型：Library file (*.lib) ， 即可选择 PCI2006.Lib 文件。该文件的路径为用户安装驱动程序后其子目录 Samples\C_Builder 下。

Visual Basic:

要使用如下函数一个关键的问题是首先必须将我们提供的模块文件(*.Bas)加入到您的 VB 工程中。其方法是选择 VB 编程环境中的工程(Project)菜单，执行其中的"添加模块"(Add Module)命令，在弹出的对话中选择 PCI2006.Bas 模块文件，该文件的路径为用户安装驱动程序后其子目录 Samples\VB 下面。

请注意，因考虑 Visual C++和 Visual Basic 两种语言的兼容问题，在下列函数说明和示范程序中，所举的 Visual Basic 程序均是需要编译后在独立环境中运行。所以用户若在解释环境中运行这些代码，我们不能保证完全顺利运行。

Delphi:

要使用如下函数一个关键的问题是首先必须将我们提供的单元模块文件（*.Pas）加入到您的 Delphi 工程中。其方法是选择 Delphi 编程环境中的 View 菜单，执行其中的"Project Manager"命令，在弹出的对话中选择*.exe 项目，再单击鼠标右键，最后 Add 指令，即可将 PCI2006.Pas 单元模块文件加入到工程中。或者在 Delphi 的编程环境中的 Project 菜单中，执行 Add To Project 命令，然后选择*.Pas 文件类型也能实现单元模块文件的添加。该文件的路径为用户安装驱动程序后其子目录 Samples\Delphi 下面。最后请在使用驱动程序接口的源程序文件中的头部的 Uses 关键字后面的项目中加入：“PCI2006”。如：

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  PCI2006;   // 注意： 在此加入驱动程序接口单元 PCI2006
```

LabVIEW/CVI :

LabVIEW 是美国国家仪器公司(National Instrument)推出的一种基于图形开发、调试和运行程序的集成化环境，是目前国际上唯一的编译型的图形化编程语言。在以 PC 机为基础的测量和工控软件中，LabVIEW 的市场普及率仅次于 C++/C 语言。LabVIEW 开发环境具有一系列优点，从其流程图式的编程、不需预先编译就存在的语法检查、调试过程使用的数据探针，到其丰富的函数功能、数值分析、信号处理和设备驱动等功能，都令人称道。关于 LabView/CVI 的进一步介绍请见本文最后一部分关于 LabView 的专述。其驱动程序接口单元模块的使用方法如下：

一、在 LabView 中打开 PCI2006.VI 文件，用鼠标单击接口单元图标，比如 CreateDevice 图标 CreateDevice 然后按 Ctrl+C 或选择 LabView 菜单 Edit 中的 Copy 命令，接着进入用户的应用程序 LabView 中，按

Ctrl+V 或选择 LabView 菜单 Edit 中的 Paste 命令，即可将接口单元加入到用户工程中，然后按以下函数原型说明或演示程序的说明连接该接口模块即可顺利使用。

二、根据LabView语言本身的规定，接口单元图标以黑色的较粗的中间线为中心，以左边的方格为数据输入端，右边的方格为数据的输出端，如ReadDeviceProAD_ NotEmpty接口单元，设备对象句柄、用户分配的数据缓冲区、要求采集的数据长度等信息从接口单元左边输入端进入单元，待单元接口被执行后，需要返回给用户的数据从接口单元右边的输出端输出，其他接口完全同理。

三、在单元接口图标中，凡标有“I32”为有符号长整型 32 位数据类型，“U16”为无符号短整型 16 位数据类型，“［U16］”为无符号 16 位短整型数组或缓冲区或指针，“［U32］”与“[U16]”同理，只是位数不一样。

第二节、设备对象管理函数原型说明

♦ 创建设备对象函数（逻辑号）

函数原型：

Visual C++ & C++Builder：

HANDLE CreateDevice (int DeviceID = 0)

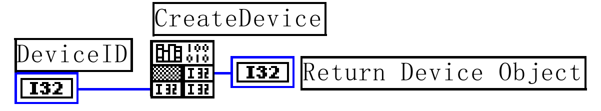
Visual Basic：

Declare Function CreateDevice Lib "PCI2006" (ByVal DeviceID As Integer = 0) As Long

Delphi：

Function CreateDevice(DeviceID : Integer = 0) : Integer;
StdCall; External 'PCI2006' Name ' CreateDevice ';

LabVIEW：



功能：该函数使用逻辑号创建设备对象，并返回其设备对象句柄 hDevice。只有成功获取 hDevice，您才能实现对该设备所有功能的访问。

参数：

DeviceID 设备 ID 标识号。

返回值：如果执行成功，则返回设备对象句柄；如果没有成功，则返回错误码 INVALID_HANDLE_VALUE。由于此函数已带容错处理，即若出错，它会自动弹出一个对话框告诉您出错的原因。您只需要对此函数的返回值作一个条件处理即可，别的任何事情您都不必做。

相关函数： CreateDevice GetDeviceCount GetDeviceCurrentID ListDeviceDlg ReleaseDevice

Visual C++ & C++Builder 程序举例：

```
    :
HANDLE hDevice;    // 定义设备对象句柄
int DeviceLgcID = 0;
hDevice = CreateDevice ( DeviceLgcID ); // 创建设备对象,并取得设备对象句柄
if(hDevice == INVALIDE_HANDLE_VALUE); // 判断设备对象句柄是否有效
{
    return;    // 退出该函数
}
    :
```

Visual Basic 程序举例：

```
    :
Dim hDevice As Long   ' 定义设备对象句柄
Dim DeviceLgcID As Long
DeviceLgcID = 0
hDevice = CreateDevice ( DeviceLgcID )  ' 创建设备对象,并取得设备对象句柄
If hDevice = INVALID_HANDLE_VALUE Then   ' 判断设备对象句柄是否有效
    MsgBox "创建设备对象失败"
    Exit Sub     ' 退出该过程
End If
    :
```

♦ **取得本计算机系统中 PCI2006 设备的总数量**

函数原型：

Visual C++ & C++Builder:

int GetDeviceCount (HANDLE hDevice)

Visual Basic:

Declare Function GetDeviceCount Lib "PCI2006" (ByVal hDevice As Long) As Integer

Delphi:

Function GetDeviceCount (hDevice : Integer) : Integer;
StdCall; External 'PCI2006' Name ' GetDeviceCount ';

LabVIEW:

GetDeviceCount
hDevice I32 → I32 Return Value

功能：取得 PCI2006 设备的数量。
参数：hDevice设备对象句柄，它应由CreateDevice创建。
返回值：返回系统中 PCI2006 的数量。
相关函数： CreateDevice　GetDeviceCount　GetDeviceCurrentID
ListDeviceDlg　ReleaseDevice

- **取得该设备当前 ID**

函数原型：

Visual C++ & C++Builder:

int GetDeviceCurrentID (HANDLE hDevice)

Visual Basic:

Declare Function GetDeviceCurrentID Lib "PCI2006" (ByVal hDevice As Long)As Integer

Delphi:

Function GetDeviceCurrentID (hDevice : Integer): Integer;
StdCall; External 'PCI2006' Name 'GetDeviceCurrentID';

LabVIEW:

请参考相关演示程序。

功能：取得 PCI2006 设备的数量。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
返回值：返回系统中当前设备相应的 ID 号。
相关函数： CreateDevice　GetDeviceCount　GetDeviceCurrentID
ListDeviceDlg　ReleaseDevice

- **用对话框控件列表计算机系统中所有 PCI2006 设备各种配置信息**

函数原型：

Visual C++ & C++Builder:

BOOL ListDeviceDlg (HANDLE hDevice)

Visual Basic:

Declare Function ListDeviceDlg Lib "PCI2006" (ByVal hDevice As Long) As Boolean

Delphi:

Function ListDeviceDlg (hDevice : Integer) : Boolean;
StdCall; External 'PCI2006' Name ' ListDeviceDlg ';

LabVIEW:

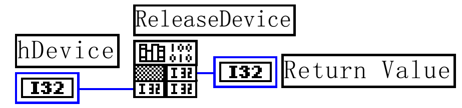
请参考相关演示程序。

功能：列表系统中 PCI2006 的硬件配置信息。
参数：hDevice设备对象句柄，它应由CreateDevice创建。
返回值：若成功，则弹出对话框控件列表所有 PCI2006 设备的配置情况。
相关函数：CreateDevice　ReleaseDevice

- **释放设备对象所占的系统资源及设备对象**

函数原型：

Visual C++ & C++Builder:
BOOL ReleaseDevice(HANDLE hDevice)
Visual Basic:
Declare Function ReleaseDevice Lib "PCI2006" (ByVal hDevice As Long) As Boolean
Delphi：
Function ReleaseDevice(hDevice : Integer) : Boolean;
StdCall; External 'PCI2006' Name ' ReleaseDevice ';
LabVIEW：



功能：释放设备对象所占用的系统资源及设备对象自身。
参数：hDevice设备对象句柄，它应由CreateDevice创建。
返回值：若成功，则返回TRUE， 否则返回FALSE， 用户可以用GetLastErrorEx捕获错误码。
相关函数：CreateDevice

应注意的是，CreateDevice必须和ReleaseDevice函数一一对应，即当您执行了一次CreateDevice后，再一次执行这些函数前，必须执行一次ReleaseDevice函数，以释放由CreateDevice占用的系统软硬件资源，如DMA控制器、系统内存等。只有这样，当您再次调用CreateDevice函数时，那些软硬件资源才可被再次使用。

第三节、AD 程序查询方式采样操作函数原型说明

◆ **初始化 AD 设备（ Initlize device AD for program mode）**

函数原型：
Visual C++ & C++Builder:
BOOL InitDeviceProAD(HANDLE hDevice,
PPCI2006_PARA_AD pADPara)
Visual Basic:
Declare Function InitDeviceProAD Lib "PCI2006" (ByVal hDevice As Long, _
ByRef pADPara As PCI2006_PARA_AD) As Boolean
Delphi:
Function InitDeviceProAD(hDevice : Integer;
pADPara : PPCI2006_PARA_AD) : Boolean;
StdCall; External 'PCI2006' Name ' InitDeviceProAD ';
LabVIEW：
请参考相关演示程序。

功能：它负责初始化设备对象中的AD部件，为设备的操作就绪做有关准备工作，如预置AD采集通道、采样频率等。但它并不启动AD设备，若要启动AD设备，须在调用此函数之后再调用StartDeviceProAD。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
pADPara 设备对象参数结构， 它决定了设备对象的各种状态及工作方式，如AD采样通道、采样频率等。关于PCI2006_PARA_AD具体定义请参考PCI2006.h(.Bas或.Pas或.VI)驱动接口文件及本文档中的《AD硬件参数结构》章节。
返回值：如果初始化设备对象成功，则返回TRUE， 否则返回FALSE， 用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数：	CreateDevice	SetDevFrequenceAD	InitDeviceProAD
	StartDeviceProAD	ReadDeviceProAD_NotEmpty	GetDevStatusProAD
	ReadDeviceProAD_Half	StopDeviceProAD	ReleaseDeviceProAD
	ReleaseDevice		

◆ **启动 AD 设备(Start device AD for program mode)**

函数原型：
Visual C++ & C++Builder:
BOOL StartDeviceProAD (HANDLE hDevice)
Visual Basic:

Declare Function StartDeviceProAD Lib "PCI2006" (ByVal hDevice As Long) As Boolean

Delphi:

```
Function StartDeviceProAD (hDevice : Integer ): Boolean;
                    StdCall; External 'PCI2006' Name ' StartDeviceProAD ';
```

LabVIEW:

请参考相关演示程序。

功能：启动AD设备，它必须在调用InitDeviceProAD后才能调用此函数。该函数除了启动AD设备开始转换以外，不改变设备的其他任何状态。

参数：hDevice设备对象句柄，它应由CreateDevice创建。

返回值：如果调用成功，则返回TRUE，且AD立刻开始转换，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数：

CreateDevice	SetDevFrequenceAD	InitDeviceProAD
StartDeviceProAD	ReadDeviceProAD_NotEmpty	GetDevStatusProAD
ReadDeviceProAD_Half	StopDeviceProAD	ReleaseDeviceProAD
ReleaseDevice		

♦ 动态改变采样频率(**Set device AD frequency**)

函数原型：

Visual C++ & C++Builder:

```
BOOL SetDevFrequenceAD (HANDLE hDevice,
                        DWORD nADFrequency)
```

Visual Basic:

```
Declare Function SetDevFrequenceAD Lib "PCI2006" (ByVal hDevice As Long, _
                                        ByVal nADFrequency As Long) As Boolean
```

Delphi:

```
Function SetDevFrequenceAD (hDevice : Integer;
                        nADFrequency: LongWord) : Boolean;
                        StdCall; External 'PCI2006' Name ' SetDevFrequenceAD ';
```

LabVIEW：

请参考相关演示程序。

功能：在AD采样过程中，可动态改变采样频率(在分组采样中只能改变组内频率Frequency)。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

nADFrequency 指定 AD 的当前采样频率。本设备的频率取值范围为[39Hz, 400KHz]。

返回值：如果调用成功，则返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数：

CreateDevice	SetDevFrequenceAD	InitDeviceProAD
StartDeviceProAD	ReadDeviceProAD_NotEmpty	GetDevStatusProAD
ReadDeviceProAD_Half	StopDeviceProAD	ReleaseDeviceProAD
ReleaseDevice		

♦ 读取 **PCI** 设备上的 **AD** 数据

① 使用 FIFO 的非空标志读取 AD 数据

函数原型：

Visual C++ & C++Builder:

```
LONG ReadDeviceProAD_NotEmpty ( HANDLE hDevice,
                                PSHORT pADBuffer,
                                ULONG nReadSizeWords,
                                BOOL bCheckOverflow = FALSE)
```

Visual Basic:

```
Declare Function ReadDeviceProAD_NotEmpty Lib "PCI2006" (ByVal hDevice As Long, _
                                        ByRef pADBuffer As Integer,_
                                        ByVal nReadSizeWords As Long,_
                                        ByVal bCheckOverflow As Boolean = False) As Long
```

Delphi:

```
Function ReadDeviceProAD_NotEmpty (hDevice : Integer;
```

pADBuffer: Pointer;
nReadSizeWords : LongWord;
bCheckOverflow: Boolean = False) : LongInt;
StdCall; External 'PCI2006' Name ' ReadDeviceProAD_NotEmpty ';

LabVIEW:
请参考相关演示程序。

功能：一旦用户使用StartDeviceProAD后，应立即用此函数读取设备上的AD数据。此函数使用FIFO的非空标志进行读取AD数据。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pADBuffer接受AD数据的用户缓冲区，它可以是一个用户定义的数组。关于如何将这些AD数据转换成相应的电压值，请参考《数据格式转换与排列规则》。

nReadSizeWords 指定一次ReadDeviceProAD_NotEmpty操作应读取多少字数据到用户缓冲区。注意此参数的值不能大于用户缓冲区ADBuffer的最大空间。此参数值只与ADBuffer[]指定的缓冲区大小有效，而与FIFO存储器大小无效。

bCheckOverflow 指定在读取 AD 数据的过程中，是否对溢出标志进行检测。默认值为 FALSE。若指定为TRUE，表示对溢出标志进行监控，若发生溢出，则该函数立即返回，其返回值表示在溢出前已成功读取的 AD数据点数，但这个返回值必将小于 nReadSizeWords 参数的值。若指定为 FALSE，则表示不对 FIFO 存储器的溢出标志进行监控，即便溢出已发生，也始终返回由 nReadSizeWords 参数指定长度的数据，其返回值也必将等于 nReadSizeWords 参数值，除非用户在这个函数返回前，就提前调用了 ReleaseDeviceProAD 函数要释放 AD设备，那么返回值也可能小于 nReadSizeWords 参数值。究竟是溢出还是提前释放 AD 引起的返回值小于nReadSizeWords 参数值，用户可以在这种情况下，调用 GetLastError 来判断。

返回值：其返回值表示所成功读取的数据点数(字)，也表示当前读操作在ADBuffer缓冲区中的有效数据量。通常情况下其返回值应与ReadSizeWords参数指定量的数据长度(字)相等，除非用户在这个读操作以外的其他线程中执行了ReleaseDeviceProAD函数中断了读操作，否则设备可能有问题。对于返回值不等于nReadSizeWords参数值的，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

注释：此函数也可用于单点读取和几个点的读取，只需要将nReadSizeWords设置成 1 或相应值即可。其使用方法请参考《高速大容量、连续不间断数据采集及存盘技术详解》章节。

相关函数：	CreateDevice	SetDevFrequenceAD	InitDeviceProAD
	StartDeviceProAD	ReadDeviceProAD_ NotEmpty	GetDevStatusProAD
	ReadDeviceProAD_Half	StopDeviceProAD	ReleaseDeviceProAD
	ReleaseDevice		

② 使用 FIFO 的半满标志读取 AD 数据

◆ **取得 FIFO 的状态标志**

函数原型：

Visual C++ & C++Builder:
```
BOOL GetDevStatusProAD ( HANDLE hDevice,
                         PBOOL bNotEmpty,
                         PBOOL bHalf,
                         PBOOL bOverflow )
```
Visual Basic:
```
Declare Function GetDevStatusProAD Lib "PCI2006" (ByVal hDevice As Long,_
                                  ByRef bNotEmpty As Boolean,_
                                  ByRef bHalf As Boolean,_
                                  ByRef bOverflow As Boolean) As Boolean
```
Delphi:
```
Function GetDevStatusProAD (hDevice : Integer;
                          bNotEmpty: Pointer;
                          bHalf : Pointer;
                          bOverflow : Pointer):Boolean;
                          StdCall; External 'PCI2006' Name ' GetDevStatusProAD ';
```
LabVIEW:
请参考相关演示程序。

功能：一旦用户使用StartDeviceProAD后，应立即用此函数查询FIFO存储器的状态（半满标志、非空标志、溢出标志）。我们通常用半满标志去同步半满读操作。当半满标志有效时，再紧接着用ReadDeviceProAD_Half读取FIFO中的半满有效AD数据。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

bNotEmpty 它将带回 FIFO 的非空状态标志。若等于 TRUE，则表示非空状态有效，否则为无效。

bHalf 它将带回 FIFO 的半满状态标志。若等于 TRUE，则表示半满状态有效，否则为无效。

bOverflow 它将带回 FIFO 的溢出状态标志。若等于 TRUE，则表示溢出状态有效，否则为无效。

返回值：若调用成功则返回TRUE，否则返回FALSE，用户可以调用GetLastErrorEx函数取得当前错误码。若用户选择半满查询方式读取AD数据，则当GetDevStatusProAD函数取得的bHalf等于TRUE，应立即调用ReadDeviceProAD_Half读取FIFO中的半满数据。否则用户应继续循环轮询FIFO半满状态，直到有效为止。注意在循环轮询期间，可以用Sleep函数抛出一定时间给其他应用程序(包括本应用程序的主程序和其他子线程)，以提高系统的整体数据处理效率。

其使用方法请参考本文档的《高速大容量、连续不间断数据采集及存盘技术详解》章节。

相关函数：	CreateDevice	SetDevFrequenceAD	InitDeviceProAD
	StartDeviceProAD	ReadDeviceProAD_NotEmpty	GetDevStatusProAD
	ReadDeviceProAD_Half	StopDeviceProAD	ReleaseDeviceProAD
	ReleaseDevice		

♦ **当 FIFO 半满信号有效时，批量读取 AD 数据**

函数原型：

Visual C++ & C++Builder:

```
BOOL ReadDeviceProAD_Half( HANDLE hDevice,
                           PSHORT pADBuffer,
                           ULONG nReadSizeWords)
```

Visual Basic:

```
Declare Function ReadDeviceProAD_Half Lib "PCI2006" ( ByVal hDevice As Long, _
                                                       ByRef pADBuffer As Integer,_
                                                       ByVal nReadSizeWords As Long) As Boolean
```

Delphi:

```
Function ReadDeviceProAD_Half(hDevice : Integer;
                               pADBuffer: Pointer;
                               nReadSizeWords : LongWord) : Boolean;
                               StdCall; External 'PCI2006' Name ' ReadDeviceProAD_Half ';
```

LabVIEW:

请参考相关演示程序。

功能：一旦用户使用GetDevStatusProAD后取得的FIFO状态bHalf等于TRUE(即半满状态有效)时，应立即用此函数读取设备上FIFO中的半满AD数据。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pADBuffer接受AD数据的用户缓冲区，通常可以是一个用户定义的数组。关于如何将这些AD数据转换成相应的电压值，请参考《数据格式转换与排列规则》。

nReadSizeWords 指定一次ReadDeviceProAD_Half操作应读取多少字数据到用户缓冲区。注意此参数的值不能大于用户缓冲区ADBuffer的最大空间，而且应等于FIFO总容量的二分之一(如果用户有特殊需要可以小于FIFO的二分之一长)。比如设备上配置了 1K FIFO，即 1024 字，那么这个参数应指定为 512 或小于 512。

返回值：如果成功的读取由nReadSizeWords参数指定量的AD数据到用户缓冲区，则返回TRUE，否则返回FALSE， 用户可用GetLastErrorEx捕获当前错误码，并加以分析。

其使用方法请参考《高速大容量、连续不间断数据采集及存盘技术详解》。

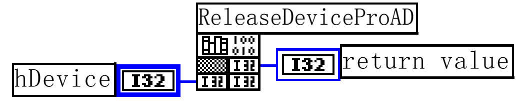
相关函数：	CreateDevice	SetDevFrequenceAD	InitDeviceProAD
	StartDeviceProAD	ReadDeviceProAD_NotEmpty	GetDevStatusProAD
	ReadDeviceProAD_Half	StopDeviceProAD	ReleaseDeviceProAD
	ReleaseDevice		

♦ **暂停 AD 设备**

函数原型：
Visual C++ & C++Builder:
BOOL StopDeviceProAD (HANDLE hDevice)
Visual Basic:
Declare Function StopDeviceProAD Lib "PCI2006" (ByVal hDevice As Long)As Boolean
Delphi:
Function StopDeviceProAD (hDevice : Integer) : Boolean;
StdCall; External 'PCI2006' Name ' StopDeviceProAD ';
LabVIEW
请参考相关演示程序。

功能：暂停AD设备。它必须在调用StartDeviceProAD后才能调用此函数。该函数除了停止AD设备不再转换以外，不改变设备的其他任何状态。此后您可再调用StartDeviceProAD函数重新启动AD，此时AD会按照暂停以前的状态（如FIFO存储器位置、通道位置）开始转换。
参数：hDevice设备对象句柄，它应由CreateDevice创建。
返回值：如果调用成功，则返回TRUE，且AD立刻停止转换，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。
相关函数： CreateDevice SetDevFrequenceAD InitDeviceProAD StartDeviceProAD ReadDeviceProAD_NotEmpty GetDevStatusProAD ReadDeviceProAD_Half StopDeviceProAD ReleaseDeviceProAD ReleaseDevice

- **释放设备上的 AD 部件**

函数原型：
Visual C++ & C++ Builder:
BOOL ReleaseDeviceProAD(HANDLE hDevice)
Visual Basic:
Declare Function ReleaseDeviceProAD Lib "PCI2006" (ByVal hDevice As Long) As Boolean
Delphi:
Function ReleaseDeviceProAD (hDevice : Integer) : Boolean;
StdCall; External 'PCI2006' Name ' ReleaseDeviceProAD ';
LabVIEW:


功能：释放设备上的 AD 部件。
参数：hDevice设备对象句柄，它应由CreateDevice创建。
返回值：若成功，则返回TRUE， 否则返回FALSE， 用户可以用GetLastErrorEx捕获错误码。

应注意的是，InitDeviceProAD必须和ReleaseDeviceProAD函数一一对应，即当您执行了一次InitDeviceProAD后，再一次执行这些函数前，必须执行一次ReleaseDeviceProAD函数，以释放由InitDeviceProAD占用的系统软硬件资源，如映射寄存器地址、系统内存等。只有这样，当您再次调用InitDeviceProAD函数时，那些软硬件资源才可被再次使用。
相关函数： CreateDevice InitDeviceProAD ReleaseDeviceProAD ReleaseDevice

- **程序查询方式采样函数一般调用顺序**

非空查询方式：
① CreateDevice
② InitDeviceProAD
③ StartDeviceProAD
④ ReadDeviceProAD_NotEmpty
⑤ StopDeviceProAD
⑥ ReleaseDeviceProAD
⑦ ReleaseDevice
注明：用户可以反复执行第④步，以实现高速连续不间断大容量采集。

半满查询方式：
① CreateDevice
② InitDeviceProAD
③ StartDeviceProAD
④ GetDevStatusProAD
⑤ ReadDeviceProAD_Half
⑥ StopDeviceProAD
⑦ ReleaseDeviceProAD
⑧ ReleaseDevice
注明：用户可以反复执行第④、⑤步，以实现高速连续不间断大容量采集。
关于两个过程的图形说明请参考《使用纲要》。

第四节、AD 中断方式采样操作函数原型说明

♦ **初始化设备上的 AD 对象**

函数原型：

Visual C++ & C++ Builder:

```
BOOL InitDeviceIntAD(HANDLE hDevice,
                     HANDLE hEvent,
                     ULONG nFifoHalfLength,
                     PPCI2006_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function InitDeviceIntAD Lib "PCI2006" (ByVal hDevice As Long, _
                                              ByVal hEvent As Long, _
                                              ByVal nFifoHalfLength As Long,_
                                              ByRef pADPara As PCI2006_PARA_AD ) As Boolean
```

Delphi:

```
Function InitDeviceIntAD (hDevice : Integer;
                          hEvent : Integer;
                          nFifoHalfLength : LongWord;
                          pADPara: PPCI2006_PARA_AD) : Boolean;
                          StdCall; External 'PCI2006' Name ' InitDeviceIntAD';
```

LabVIEW:

请参考相关演示程序。

功能：它负责初始化设备对象中的AD部件，为设备操作就绪有关工作，如预置AD采集通道，采样频率等。且让设备上的AD部件以硬件中断的方式工作，其中断源信号由FIFO芯片半满管脚提供。但它不启动AD采样，那么需要在此函数被成功调用之后，再调用StartDeviceIntAD函数即可启动AD采样。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

hEvent 中断事件对象句柄，它应由CreateSystemEvent函数创建。它被创建时是一个不发信号且自动复位的内核系统事件对象。当硬件中断发生，这个内核系统事件被触发。用户应在数据采集子线程中使用WaitForSingleObject这个Win32 函数来接管这个内核系统事件。当中断没有到来时，WaitForSingleObject将使所在线程进入睡眠状态，此时，它不同于程序轮询方式，它并不消耗CPU时间。当hEvent事件被触发成发信号状态，那么WaitForSingleObject将唤醒所在线程，可以工作了，比如取FIFO中的数据、分析数据等，且复位该内核系统事件对象，使其处于不发信号状态，以便在取完FIFO数据等工作后，让所在线程再次进入睡眠状态。所以利用中断方式采集数据，其效率是最高的。其具体实现方法请参考《高速大容量、连续不间断数据采集及存盘技术详解》。

nFifoHalfLength 告诉设备对象，FIFO 存储器半满长度大小。该参数很关键，因为不仅决定了设备对象每次产生半满中断时应读入 AD 数据的点数，同时，它也决定了一级缓冲队列中每个元素对应的缓冲区大小。比如，nFifoHalfLength 等于 2048，则设备对象在系统空间中建立具有 64 个元素，且每个元素对应于 2048 个字长且物理连续的一级缓冲队列。但是该参数可以根据用户特殊需要，将其置成小于 FIFO 存储器实际的半满长度的值。比如用户要求在频率一定的情况下，提高 FIFO 半满中断事件的频率等，那么可以将此参数置成小于 FIFO 半满长度的值，但是绝不能大小半满长度。在工作期间，此队列的维护和管理完全由设备对象管理，与用户无关，用户只需要用 ReadDeviceIntAD 函数简单地读取 AD 数据，并注意检查其返回值即可。

pADPara设备对象参数结构指针， 它的各成员值决定了设备上的AD对象的各种状态及工作方式，如AD采样通道、采样频率等。请参考《硬件参数结构》章节。

返回值：如果初始化设备对象成功，则返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数： CreateDevice InitDeviceIntAD StartDeviceIntAD ReadDeviceIntAD StopDeviceIntAD ReleaseDeviceIntAD ReleaseDevice

◆ 启动设备上的 **AD** 部件

函数原型：

Visual C++ & C++ Builder:

```
BOOL StartDeviceIntAD (HANDLE hDevice)
```

Visual Basic:

```
Declare Function StartDeviceIntAD Lib "PCI2006" (ByVal hDevice As Long ) As Boolean
```

Delphi:

```
Function StartDeviceIntAD (hDevice : Integer) : Boolean;
                    StdCall; External 'PCI2006' Name ' StartDeviceIntAD';
```

LabVIEW:

请参考相关演示程序。

功能：在InitDeviceIntAD被成功调用之后，调用此函数即可启动设备上的AD部件，让设备开始AD采样。

参数：hDevice设备对象句柄，它应由CreateDevice创建。

返回值：若成功，则返回TRUE，意味着AD被启动，否则返回FALSE，用户可以用GetLastErrorEx捕获错误码。

相关函数： CreateDevice InitDeviceIntAD StartDeviceIntAD ReadDeviceIntAD StopDeviceIntAD ReleaseDeviceIntAD ReleaseDevice

◆读取 **PCI** 设备上的 **AD** 数据

Visual C++ & C++Builder:

```
DWORD ReadDeviceIntAD (HANDLE hDevice,
                    PSHORT pADBuffer,
                    ULONG ReadSizeWords)
```

Visual Basic:

```
Declare Function ReadDeviceIntAD Lib "PCI2006" (ByVal hDevice As Long,_
                                        ByRef pADBuffer As Integer,_
                                        ByVal ReadSizeWords As Long) As Long
```

Delphi:

```
Function ReadDeviceIntAD (hDevice : Integer;
                    pADBuffer: Pointer;
                    ReadSizeWords : LongWord) : LongWord;
                    StdCall; External 'PCI2006' Name ' ReadDeviceIntAD';
```

LabVIEW:

请参考相关演示程序。

功能：户在半满中断事件的同步下，读取半满 AD 数据。

一旦用户使用StartDeviceIntAD后，应立即用WaitForSingleObject等待中断事件hIntEvent的发生，如果FIFO还没有达到半满状态，即中断事件还发生，则数据采集线程WaitForSingleObject的作用下自动进入睡眠状态(此状态下，数据采集线程或代码不消耗CPU时间)。 当中断事件发生时线程被突发唤醒，即在WaitForSingleObject后的代码将被立即得到执行，因此为了提高数据吞吐率，在WaitForSingleObject之后，应紧接着用ReadDeviceIntAD函数读取FIFO半满数据。注意看演示程序如何处理这个问题。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pADBuffer接受AD数据的用户缓冲区，可以是一个相应类型的足够大的数组，也可以是用户使用内存分配函数分配的内存空间。关于如何将缓冲区中的这些AD数据转换成相应的电压值，请参考《数据格式转换与排列规则》。

ReadSizeWords 指定一次ReadDeviceIntAD操作应读取多少字数据到用户缓冲区。注意此参数的值不能大于用户缓冲区pADBuffer的最大空间长度，且由于是半满读取数据，所以这个参数必须等于板上FIFO存储器总容量的二分之一，比如FIFO为 1K长度（即 1024 点），则此参数应为 512，若为 4K（即 4096 点）长度，则此参数应为 2048，其他情况以此类推。当然特殊情况下，比如用户不要求数据连续或不担心丢点问题，则可以将此参数设得比FIFO存储器的半满长度小。需要用户特别注意的是此参数必须与InitDeviceIntAD函数中的nFifoHalfLength参数相等，才能实现连续数据采集。如果大于nFifoHalfLength，此会造成缓冲访问异常，严重时可能会使整过Windows系统崩溃，如果小于nFifoHalfLength，则会丢失n个点的数据在一级缓冲内(n为nFifoHalfLength 减去 nReadSizeWords的差值)。

返回值：如果失败，即一级缓冲队列溢出则返回 0xe1000000 码，如果成功，则返回一级缓冲队列中的未被ReadDeviceIntAD读空的缓冲队列元素数量。一个元素对应于一个由InitDeviceIntAD函数的nFifoHalfLength参数指定大小的系统物理缓冲区。

注释：由于设备对象在系统空间中维护两个当前指针，且这两个指针最初都指向缓冲队列中的第一个元素。为了便于说明，我们将这两个指针分别命名为：用户指针和系统指针。当每执行此函数一次，设备对象将用户指针指向的缓冲区中的数据映射到用户空间pADBuffer中，且将用户指针下移一个元素位置。而系统指针则不随用户的操作而改变。它是设备对象强制自动维护的指针。它的改变速度只与AD数据转换有关。可见，不管用户有没有读走当前指针指向的一级缓冲区中的数据，或者整个Windows系统有多忙，但这个系统指针每到一个半满状态时，它总会自动下移一个元素位置。设备对象根据某些状态信息和利用巧妙算法，统计出已经采集了但用户迟迟没有读走的缓冲区数量，这个数量便是ReadDeviceIntAD返回的正确值，且判断数据是否重叠或溢出，这个状态便是ReadDeviceIntAD返回的 0xe1000000 码。如果用户的处理速度与得到设备对象的传输速度一样，那么ReadDeviceIntAD的返回值应等于 0，如果某一次在WaitForSingleObject之后执行ReadDeviceIntAD所返回的值不为 0，且不为 0xe1000000，假如是 5，则视为一级缓冲区中的数据已有 5 个元素指向的数据是已采集的新数据，那么用户应接着用循环语句连读 5 次数据，直到ReadDeviceIntAD返回 0 为止。其他情况以此类推。此种方案的使用，让用户即便是在高速采集数据时，也能在很大程度上象往常一样随意进行窗口菜单等突发操作。

相关函数： CreateDevice　InitDeviceIntAD　StartDeviceIntAD　ReadDeviceIntAD　StopDeviceIntAD　ReleaseDeviceIntAD　ReleaseDevice

◆ 暂停设备上的 AD 采样工作

函数原型：

Visual C++ & C++ Builder:

BOOL StopDeviceIntAD (HANDLE hDevice)

Visual Basic:

Declare Function StopDeviceIntAD Lib "PCI2006" (ByVal hDevice As Long) As Boolean

Delphi：

Function StopDeviceIntAD (hDevice : Integer) : Boolean;
StdCall; External 'PCI2006' Name ' StopDeviceIntAD ';

LabVIEW:

请参考相关演示程序。

功能：在StartDeviceIntAD被成功调用之后，用户可以在任何时候调用此函数停止AD采样(必须在ReleaseDeviceDmaAD之间被调用)，注意它不改变设备的其它任何状态。如果过后用户再调用StartDeviceDmaAD，那么设备会接着停止前的状态(如通道位置)继续开始正常的AD数据转换。

参数：hDevice设备对象句柄，它应由CreateDevice创建。

返回值：若成功，则返回TRUE，意味着AD被停止，否则返回FALSE，用户可以用GetLastErrorEx捕获错误码。

相关函数： CreateDevice　InitDeviceIntAD　StartDeviceIntAD　ReadDeviceIntAD　StopDeviceIntAD　ReleaseDeviceIntAD　ReleaseDevice

◆ 释放设备上的 AD 部件

函数原型：

Visual C++ & C++ Builder:

BOOL ReleaseDeviceIntAD (HANDLE hDevice)

Visual Basic:

Declare Function ReleaseDeviceIntAD Lib "PCI2006" (ByVal hDevice As Long) As Boolean

Delphi：

Function ReleaseDeviceIntAD (hDevice : Integer) : Boolean;
StdCall; External 'PCI2006' Name ' ReleaseDeviceIntAD ';

LabVIEW:

请参考相关演示程序。

功能：释放设备上的AD部件，如果AD没有被StopDeviceIntAD函数停止，则此函数在释放AD部件之前先停止AD部件。

参数：hDevice设备对象句柄，它应由CreateDevice创建。

返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用GetLastErrorEx捕获错误码。

相关函数： CreateDevice InitDeviceIntAD StartDeviceIntAD
ReadDeviceIntAD StopDeviceIntAD ReleaseDeviceIntAD
ReleaseDevice

应注意的是，InitDeviceIntAD必须和ReleaseDeviceIntAD函数一一对应，即当您执行了一次InitDeviceIntAD后，再一次执行这些函数前，必须执行一次ReleaseDeviceIntAD函数，以释放先前由InitDeviceIntAD占用的系统软硬件资源，如映射寄存器地址、系统内存等。只有这样，当您再次调用InitDeviceIntAD函数时，那些软硬件资源才可被再次使用。

◆ 函数一般调用顺序

① CreateDevice
② CreateSystemEvent(公共函数)
③ InitDeviceIntAD
④ StartDeviceIntAD
⑤ WaitForSingleObject(WIN32 API 函数，详细说明请参考 MSDN 文档)
⑥ ReadDeviceIntAD
⑦ StopDeviceIntAD
⑧ ReleaseDeviceDmaAD
⑨ ReleaseSystemEvent（公共函数）
⑩ ReleaseDevice

注明：用户可以反复执行第⑤⑥⑦步，以实现高速连续不间断大容量采集。

关于这个过程的图形说明请参考《使用纲要》。

第五节、AD 硬件参数保存与读取函数原型说明

◆ **从 Windows 系统中读入硬件参数函数**

函数原型：

Visual C++ & C++ Builder:

BOOL LoadParaAD(HANDLE hDevice,
PPCI2006_PARA_AD pADPara)

Visual Basic:

Declare Function LoadParaAD Lib "PCI2006" (ByVal hDevice As Long, _
ByRef pADPara As PCI2006_PARA_AD) As Boolean

Delphi:

Function LoadParaAD (hDevice : Integer;
pADPara: PPCI2006_PARA_AD) : Boolean;
StdCall; External 'PCI2006' Name ' LoadParaAD ';

LabVIEW:

请参考相关演示程序。

功能：负责从 Windows 系统中读取设备的硬件参数。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pADPara属于PPCI2006_PARA_AD的结构指针类型，它负责返回PCI硬件参数值，关于结构指针类型PPCI2006_PARA_AD请参考PCI2006.h或PCI2006.Bas或PCI2006.Pas函数原型定义文件，也可参考本文《硬件参

数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaAD SaveParaAD ReleaseDevice

◆ 往 **Windows** 系统写入设备硬件参数函数

函数原型：

Visual C++ & C++ Builder:

```
BOOL SaveParaAD (HANDLE hDevice,
                 PPCI2006_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function SaveParaAD Lib "PCI2006" (ByVal hDevice As Long, _
                                ByRef pADPara As PCI2006_PARA_AD) As Boolean
```

Delphi:

```
Function SaveParaAD (hDevice : Integer;
                     pADPara: PPCI2006_PARA_AD) : Boolean;
                     StdCall; External 'PCI2006' Name ' SaveParaAD ';
```

LabVIEW:

请参考相关演示程序。

功能：负责把用户设置的硬件参数保存在 Windows 系统中，以供下次使用。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pADPara设备硬件参数，关于PCI2006_PARA_AD的详细介绍请参考PCI2006.h或PCI2006.Bas或PCI2006.Pas函数原型定义文件，也可参考本文《硬件参数结构》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice LoadParaAD SaveParaAD ReleaseDevice

第六节、DA 模拟量输出操作函数原型说明

◆ 初始化 **DA**，如设置模拟量输出量程范围

函数原型：

Visual C++ & C++Builder:

```
BOOL InitDeviceProDA (HANDLE hDevice,
                      int ResetMode,
                      int nDAChannel)
```

Visual Basic:

```
Declare Function InitDeviceProDA Lib "PCI2006" (ByVal hDevice As Long, _
                                    ByVal ResetMode As Integer,_
                                    ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function InitDeviceProDA ( hDevice : Integer;
                           ResetMode : Integer;
                           nDAChannel : Integer) : Boolean;
                           StdCall; External 'PCI2006' Name ' InitDeviceProDA ';
```

LabVIEW:

请参考相关演示程序。

功能：设置指定通道的模拟量输出量程范围。它决定了WriteDeviceProDA函数。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

ResetMode DA 的复位方式。

常量名	常量值	功能定义
PCI2006_RESET_MODE_NEGATIVE	0x0000	负满度（-5V 或-10V...）复位
PCI2006_RESET_MODE_ZERO	0x0001	零点(0V)复位

nDAChannel DA 通道，取值为(0-1)。

返回值：若成功，返回TRUE，则可以调用StartDeviceProDA启动DA开始转换；否则返回FALSE，您可以调用GetLastErrorEx函数取得错误或错误字符信息。

相关函数:　CreateDevice　InitDeviceProDA　WriteDeviceProDA　ReleaseDevice

- **向 DA 的 FIFO 中写入批量数据（通常为 FIFO 的半满长度）**

函数原型：

Visual C++ & C++Builder:

```
BOOL WriteDeviceProDA (HANDLE hDevice,
                       WORD DAData,
                       int nDAChannel)
```

Visual Basic:

```
Declare Function WriteDeviceProDA Lib "PCI2006" ( ByVal hDevice As Long, _
                                                   ByVal DAData As Integer,_
                                                   ByVal nDAChannel As Integer) As Boolean
```

Delphi:

```
Function WriteDeviceProDA ( hDevice : Integer;
                            DAData : Word;
                            nDAChannel : Integer) : Boolean;
                            StdCall; External 'PCI2006' Name ' WriteDeviceProDA ';
```

LabVIEW:

请参考相关演示程序。

功能　向 DA 的 FIFO 中写入批量数据（通常为 FIFO 的半满长度）。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

DAData准备输出的DA数据LSB原码，关于如何将电压数据转换成相应Lsb原码，请参考《DA电压值转换成LSB原码数据的换算方法》章节。

nDAChannel DA 通道，取值为(0-1)。

返回值：若成功，则返回 TRUE，否则返回 FALSE，　用户可以用 GetLastError 捕获错误码。

相关函数:　CreateDevice　InitDeviceProDA　WriteDeviceProDA　ReleaseDevice

- 以上函数调用一般顺序

① CreateDevice

② InitDeviceProDA

③ WriteDeviceProDA（往FIFO存储器中写入全满数据）

第七节、DIO 数字量输入输出开关量操作函数原型说明

- 开关量输入

函数原型：

Visual C++ & C++Builder:

```
BOOL GetDeviceDI ( HANDLE hDevice,
                   PPCI2006_PARA_DI pDIPara)
```

Visual Basic:

```
Declare Function GetDeviceDI Lib "PCI2006" ( ByVal hDevice As Long, _
                                             ByVal pDIParaAs PCI2006_PARA_DI) As Boolean
```

Delphi:

```
Function GetDeviceDI ( hDevice : Integer;
                       pDIPara: PCI2006_PARA_DI) : Boolean;
                       StdCall; External 'PCI2006' Name ' GetDeviceDI ';
```

LabVIEW:

请参考相关演示程序。

功能：负责将 PCI 设备上的输入开关量状态读入内存。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pDIPara 十六路开关量输入状态的参数结构，共有 16 个元素，分别对应于 DI0-DI15 路开关量输入状态位。如果 bDISts[0]等于“1”则表示 0 通道处于开状态，若为“0”则 0 通道为关状态。其他同理。

返回值：若成功，返回 TRUE，其 pPara 中的值有效；否则返回 FALSE，其 pPara 中的值无效。

相关函数：　CreateDevice　SetDeviceDO　ReleaseDevice

- 开关量输出

函数原型：

Visual C++ & C++Builder:

```
BOOL SetDeviceDO (HANDLE hDevice,
                  PPCI2006_PARA_DO pDOPara)
```

Visual Basic:

```
Declare Function SetDeviceDO Lib "PCI2006" (ByVal hDevice As Long, _
                                    ByVal pDOParaAs PCI2006_PARA_DO) As Boolean
```

Delphi:

```
Function SetDeviceDO (hDevice : Integer;
                      pDOPara:PPCI2006_PARA_DO) : Boolean;
                      StdCall; External 'PCI2006' Name ' SetDeviceDO ';
```

LabVIEW:

请参考相关演示程序。

功能：负责将 PCI 设备上的输出开关量置成相应的状态。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pDOPara 十六路开关量输出状态的参数结构，共有 16 个成员变量，分别对应于 DO0-DO15 路开关量输出状态位。比如置 pPara->DO0 为“1”则使 0 通道处于“开”状态，若为“0”则置 0 通道为“关”状态。其他同理。请注意，在实际执行这个函数之前，必须对这个参数结构的 DO0 至 DO15 共 16 个成员变量赋初值，其值必须为“1”或“0”。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数：　CreateDevice　GetDeviceDI　ReleaseDevice

- 以上函数调用一般顺序

① CreateDevice

② SetDeviceDO(或GetDeviceDI，当然这两个函数也可同时进行)

③ ReleaseDevice

用户可以反复执行第②步，以进行数字 I/O 的输入输出（数字 I/O 的输入输出及 AD 采样可以同时进行，互不影响）。

第四章　硬件参数结构

第一节、AD 硬件参数结构（PCI2006_PARA_AD）

Visual C++ & C++Builder：

```
typedef struct _CHANNEL_ARRAY_AD
{
    DWORD ADChannel;        // AD 通道号
    DWORD ADGains;          // AD 增益
} CHANNEL_ARRAY_AD;

typedef struct _PCI2006_PARA_AD
{
    DWORD ADMode;                    // 连续采集和分组采集方式选择
    DWORD ChannelCount;              // 通道总数(1-32)
    CHANNEL_ARRAY_AD ChannelArray[32]; // 采样阵列(包括通道和增益)
```

```
	DWORD Frequency;			// AD 采集频率(Hz)
	DWORD GroupInterval;		// 分组采样时，相邻组的时间间隔(uS)
	DWORD TriggerSource;		// 内触发和外触发方式选择
	DWORD OutTriggerEdge;		// 外触发上升沿和下降沿类型选择
	DWORD OutDigitAnalog;		// 外触发数字和模拟方式选择
	DWORD ClockSource;			// 允许使用外部时钟
} PCI2006_PARA_AD, *PPCI2006_PARA_AD;
```

Visual Basic：

```
Private Type CHANNEL_ARRAY_AD
	ADChannel As Long		' AD 通道号
	ADGains As Long		' AD 增益
End Type

Private Type PCI2006_PARA_AD
	ADMode As Long
	ChannelCount As Long
	ChannelArray(32) As CHANNEL_ARRAY_AD
	Frequency As Long
	GroupInterval As Long
	TriggerSource As Long
	OutTriggerEdge As Long
	OutDigitAnalog As Long
	ClockSource As Long
End Type
```

Delphi:

```
type	// 定义结构体数据类型
	PCHANNEL_ARRAY_AD = ^ CHANNEL_ARRAY_AD;	// 指针类型结构
	CHANNEL_ARRAY_AD = record		// 标记为记录型
	ADChannel: LongWord;
	ADGains: LongWord;
End;

type	// 定义结构体数据类型
	PPCI2006_PARA_AD = ^ PCI2006_PARA_AD;	// 指针类型结构
	PCI2006_PARA_AD = record		// 标记为记录型
	ADMode : LongWord;
	ChannelCount: LongWord;
	ChannelArray[32]: CHANNEL_ARRAY_AD;
	Frequency : LongWord;
	GroupInterval: LongWord;
	TriggerSource: LongWord;
	OutTriggerEdge: LongWord;
	OutDigitAnalog: LongWord;
	ClockSource: LongWord;
End;
```

LabVIEW:

请参考相关演示程序。

该结构实在太简易了，其原因就是 PCI 设备是系统全自动管理的设备，再加上驱动程序的合理设计与封装，什么端口地址、中断号等将与 PCI 设备的用户永远告别，一句话 PCI 设备是一种更易于管理和使用的设备。

此结构主要用于设定设备AD硬件参数值，用这个参数结构对设备进行硬件配置完全由InitDeviceProAD函数自动完成。用户只需要对这个结构体中的各成员简单赋值即可。

一、关于 CHANNEL_ARRAY_AD 参数结构的说明。

ADChannel AD 采样首通道号，取值范围应根据设备的总通道数设定，本设备的通道号取值为：0～15。

ADGains 硬件采样增益，其取值分别为 1、2、4、8 或 1、10、100、1000。如果 AD 转换器前使用的是 PGA202 放大器，则这些值分别表示放大倍数为 1、10、100、1000 倍，如果 AD 转换器前使用的是 PGA203 放大器，则其值分别表示放大倍数为 1、2、4、8 倍。除了 1、2、3、4 四个值以外，其它值均为非法值。

常量名	常量值	功能定义
PCI2006_1MULT_GAINS	0x0000	1 倍增益(使用 PGA202 或 PGA203 放大器)
PCI2006_10MULT_GAINS	0x0001	10 倍增益(使用 PGA202 放大器)
PCI2006_100MULT_GAINS	0x0002	100 倍增益(使用 PGA202 放大器)
PCI2006_1000MULT_GAINS	0x0003	1000 倍增益(使用 PGA202 放大器)
PCI2006_2MULT_GAINS	0x0001	2 倍增益(使用 PGA203 放大器)
PCI2006_4MULT_GAINS	0x0002	4 倍增益(使用 PGA203 放大器)
PCI2006_8MULT_GAINS	0x0003	8 倍增益(使用 PGA203 放大器)

二、关于 PCI2006_PARA_AD 参数结构说明

硬件参数说明:

ADMode AD 连续采集和分组采集方式选择，其选项如下：

常量名	常量值	功能定义
PCI2006_SEQUENCE_MODE	0x0000	连续采集
PCI2006_GROUP_MODE	0x0001	分组采集

ChannelCount AD 采样通道总数，取值范围应根据设备的总通道数设定，本设备的通道总数取值为：1～32。

ChannelArray AD 采样通道阵列，该成员变量属于数组类型，共 32 个元素，每个元素的类型属于 CHANNEL_ARRAY_AD 结构体。每个元素(即每一个 CHANNEL_ARRAY_AD)管理该元素所处位置上的通道号和该通道所使用的增益两个信息。举列如下：

第一项举例：ChannelArray[0].ADChannel = 2;　　　ChannelArray[0].ADGains = 1;

说明：在第 0 位置上采样第 2 号通道，该通道使用硬件增益 1 倍增益

第二项举例：ChannelArray[1].ADChannel = 4;　　　ChannelArray[1].ADGains = 2;

说明：在第 1 位置上采样第 4 号通道，该通道使用 2 倍增益

虽然此处只做了两项举列，但足以看得出本设备 0 号到 15 号通道是如何任意组合以实现任意通道切换的。但是需要注意的我们必须依次使用最前端的元素。比如只采集任意一个通道即 ChannelCount = 1，那么就必须使用 ChannelArray[0]元素，即该元素中的 ADChannel 和 ADGains 进行赋值初始化，如果要采集任意 2 个通道，那么就必须使用 ChannelArray[0]和 ChannelArray[1]两个元素，如果是 3 个通道，则使用 ChannelArray[0,1,2]三个元素。其他情况同理。也可以将这个阵列理解为一张表格：(注意：所有指定通道在从 0 位置切换到 ChannelCount-1 这样的过程我们称为一个采样轮回)

阵列元素号	通道号	增益
0	12	1
1	3	1
2	5	3
3	8	2
4	6	1
5	0	1
6	4	2
7	4	1
8	3	3
9	6	4
10	14	3
11	12	1
12	2	1
13	0	1
14	3	2
15	1	3
:	:	:
32	0	1

一个轮回共采集 12 个点，即 ChannelCount=12

说明：通道不仅可以任意切换，同时，某个通道可以在每个轮回中重复出现多次。每个轮回切换点数由ChannelCount 的值决定，切换的起点是这个表的表头 0 位置，终点是 ChannelCount –1 的位置。其余为无效表项。

Frequency AD 采样频率，单位 Hz，其范围应根据具体的设备而定，但其最小值为 39Hz。切忌不能等于 0，本设备的 AD 采样频率取值范围为[39, 400K]。

GroupInterval AD 分组采样时，相邻组的时间间隔(uS)。

TriggerSource AD 转换触发源，其选项如下：

常量名	常量值	功能定义
PCI2006_IN_TRIGGER	0x0000	内触发方式
PCI2006_OUT_TRIGGER	0x0001	外触发方式

若等于常量 PCI2006_IN_TRIGGER 则为内部定时触发，若等于常量 PCI2006_OUT_TRIGGER 则为外触发。两种方式的主要区别是：外触发是当设备被 InitDeviceProAD(InitDeviceIntAD)函数初始化就绪后，并没有立即启动 AD 采集，仅当外接管脚 ATR 或 DTR(在 62 芯 D 型头上)上有一个由低至高变化的上升沿(模拟信号或 TTL 电平)时，AD 转换器便被启动，且按用户预先设定的采样频率由板上的硬件定时器时定触发 AD 等间隔转换每一个 AD 数据，此后，除非用户重新初始化设备，否则，ATR 或 DTR 管脚上所产生的新的上升沿信号并不影响 AD 转换进程。因此如果用户不断的使下一个触发信号有效，那么您必须在每一个外触发信号到来之前重新初始化设备。而对于内触发方式则与 ATR、DTR 管脚上的信号无任何关系，它是在用户调用 InitDeviceProAD(InitDeviceIntAD)函数初始化设备时，由这个函数中的最后一条软件指令立即启动 AD 转换器，AD 转换器便以指定的频率由板上定时器等间隔定时触发 AD 转换。指两种触发方式的应用场合：对于瞬间变化(持续时间短、变化频率较高)、或随机性较强的信号的测量和采样。或是需要精确定位所要采集的一批 AD 数据中的第一个点的时间轴，那么您需要使用外触发方式。而对于持续变化时间较长，不需要精确定位信号起点的信号，则一般使用内触发方式。

OutTriggerEdge AD 外触发方式使用的边沿方式，其选项如下：

常量名	常量值	功能定义
PCI2006_RISING_EDGE	0x0000	外触发上升沿触发方式
PCI2006_FALLING_EDGE	0x0001	外触发下降沿触发方式

OutDigitAnalog AD 外触发数字和模拟方式选择，其选项如下：

常量名	常量值	功能定义
PCI2006_ANALOG_TRIGGER	0x0000	模拟量外触发
PCI2006_DIGIT_TRIGGER	0x0001	数字量外触发

ClockSource AD 允许使用外部时钟，其选项如下：

常量名	常量值	功能定义
PCI2006_IN_CLOCK	0x0000	内部时钟定时触发
PCI2006_OUT_CLOCK	0x0001	外部时钟定时触发

相关函数： CreateDevice LoadParaAD SaveParaAD
ReleaseDevice

第二节、数字量输入参数（PCI2006_PARA_DI）

Visual C++ & C++Builder:

```
typedef struct _PCI2006_PARA_DI      // 数字量输入参数
{
BYTE DI0;       // 0 通道
BYTE DI1;       // 1 通道
BYTE DI2;       // 2 通道
BYTE DI3;       // 3 通道
BYTE DI4;       // 4 通道
BYTE DI5;       // 5 通道
BYTE DI6;       // 6 通道
BYTE DI7;       // 7 通道
```

```
BYTE DI8;        // 8 通道
BYTE DI9;        // 9 通道
BYTE DI10;       // 10 通道
BYTE DI11;       // 11 通道
BYTE DI12;       // 12 通道
BYTE DI13;       // 13 通道
BYTE DI14;       // 14 通道
BYTE DI15;       // 15 通道
} PCI2006_PARA_DI,*PPCI2006_PARA_DI;
```

Visual Basic：

```
Type PCI2006_PARA_DI
DI0 As Byte   ' 0 通道
DI1 As Byte   ' 1 通道
DI2 As Byte   ' 2 通道
DI3 As Byte   ' 3 通道
DI4 As Byte   ' 4 通道
DI5 As Byte   ' 5 通道
DI6 As Byte   ' 6 通道
DI7 As Byte   ' 7 通道
DI8 As Byte   ' 8 通道
DI9 As Byte   ' 9 通道
DI10 As Byte   ' 10 通道
DI11 As Byte   ' 11 通道
DI12 As Byte   ' 12 通道
DI13 As Byte   ' 13 通道
DI14 As Byte   ' 14 通道
DI15 As Byte   ' 15 通道
End Type
```

Delphi:

```
Type   //  定义结构体数据类型
  PPCI2006_PARA_DI = ^PCI2006_PARA_DI;   //  指针类型结构
  PCI2006_PARA_DI = record          //  标记为记录型
    DI0: Byte;    // 0 通道
    DI1: Byte;    // 1 通道
    DI2: Byte;    // 2 通道
    DI3: Byte;    // 3 通道
    DI4: Byte;    // 4 通道
    DI5: Byte;    // 5 通道
    DI6: Byte;    // 6 通道
    DI7: Byte;    // 7 通道
    DI8: Byte;    // 8 通道
    DI9: Byte;    // 9 通道
    DI10: Byte;    // 10 通道
    DI11: Byte;    // 11 通道
    DI12: Byte;    // 12 通道
    DI13: Byte;    // 13 通道
    DI14: Byte;    // 14 通道
    DI15: Byte;    // 15 通道
End;
```

该参数结构的使用极大的方便了不熟悉硬件端口控制和二进制位操作的用户。在这里您不需要了解技术细节，只需要执行GetDeviceDI即可完成数字量输入操作。然后象Visual Basic中的属性操作那样，简单的进行属

性成员分析即可确定各路状态。

关于LabView的参数，由于需要的是返回值，因此根据LabView的特点，应分配一个 16 字节的内存单元，每一个字节的内存单元对应相应位置上的开关量输入状态。要使用这些状态，则应在GetDeviceDI之后，将存放实际的当前开关量状态的内存单元用Index Array数组操作控件将其每一路开关量状态分离出来，即可确定每一路开关输入状态。详见开关量输入输出LabView演示部分。

其每一个成员变量对应于相应的 DI 通道，即 DI0-DI15 分别对应于 DI 通道 0-15。且这些成员变量只能是“0”或“1”数值。“0”代表“关”状态或“低”状态，“1”代表“开”状态或“高”状态。

第三节、数字量输出参数（PCI2006_PARA_DO）

Visual C++ & C++Builder:

```
typedef struct _PCI2006_PARA_DO          // 数字量输出参数
{
BYTE DO0;           // 0 通道
BYTE DO1;           // 1 通道
BYTE DO2;           // 2 通道
BYTE DO3;           // 3 通道
BYTE DO4;           // 4 通道
BYTE DO5;           // 5 通道
BYTE DO6;           // 6 通道
BYTE DO7;           // 7 通道
BYTE DO8;           // 8 通道
BYTE DO9;           // 9 通道
BYTE DO10;           // 10 通道
BYTE DO11;           // 11 通道
BYTE DO12;           // 12 通道
BYTE DO13;           // 13 通道
BYTE DO14;           // 14 通道
BYTE DO15;           // 15 通道
} PCI2006_PARA_DO,*PPCI2006_PARA_DO;
```

Visual Basic：

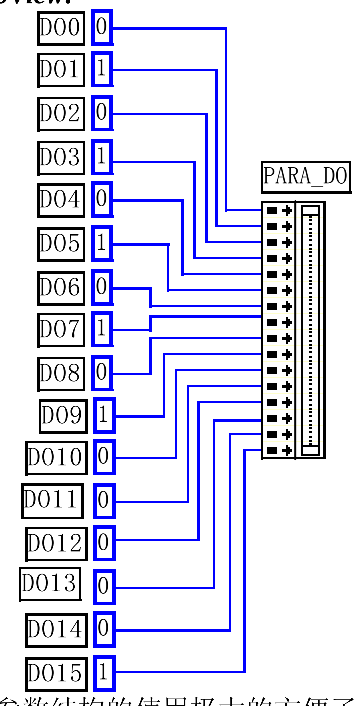
```
Type PCI2006_PARA_DO
DO0 As Byte   ‘ 0 通道
DO1 As Byte   ‘ 1 通道
DO2 As Byte   ‘ 2 通道
DO3 As Byte   ‘ 3 通道
DO4 As Byte   ‘ 4 通道
DO5 As Byte   ‘ 5 通道
DO6 As Byte   ‘ 6 通道
DO7 As Byte   ‘ 7 通道
DO8 As Byte   ‘ 8 通道
DO9 As Byte   ‘ 9 通道
DO10 As Byte   ‘ 10 通道
DO11 As Byte   ‘ 11 通道
DO12 As Byte   ‘ 12 通道
DO13 As Byte   ‘ 13 通道
DO14 As Byte   ‘ 14 通道
DO15 As Byte   ‘ 15 通道
End Type
```

Delphi:

```
  Type   // 定义结构体数据类型
  PPCI2006_PARA_DO = ^PCI2006_PARA_DO;   // 指针类型结构
```

```
    PCI2006_PARA_DO = record          //  标记为记录型
        DO0: Byte;      // 0 通道
        DO1: Byte;      // 1 通道
        DO2: Byte;      // 2 通道
        DO3: Byte;      // 3 通道
        DO4: Byte;      // 4 通道
        DO5: Byte;      // 5 通道
        DO6: Byte;      // 6 通道
        DO7: Byte;      // 7 通道
        DO8: Byte;      // 8 通道
        DO9: Byte;      // 9 通道
        DO10: Byte;     // 10 通道
        DO11: Byte;     // 11 通道
        DO12: Byte;     // 12 通道
        DO13: Byte;     // 13 通道
        DO14: Byte;     // 14 通道
        DO15: Byte;     // 15 通道
    End;
```

LabView:



该参数结构的使用极大的方便了不熟悉硬件端口控制和二进制位操作的用户。在这里您不需要了解技术细节，只需要象Visual Basic中的属性操作那样，只需要有简单的进行属性赋值，然后执行SetDeviceDO即可完成数字量输出。注意关于LabView的参数定义，他最主要表达了在LabView环境中怎样使用SetDeviceDO实现开关量输出操作的基本实现方法。在用户实际使用中，您可以将左边的常量图标换成开关控件图标等，以实现动态改变开关量输出状态。但需要注意的是开关控件图标（xxx Switch）输出的值是布尔变量，因此在开关控件图标与PPCI2006_PARA_DO之间，应使用Boolean To (0,1)逻辑转换控件，即先将布尔变量转换成 0 或 1 的整型值，再将这个整型值传递给PPCI2006_PARA_DO，详见开关量输入输出LabView演示部分。

其每一个成员变量对应于相应的 DO 通道，即 DO0-DO15 分别对应于 DO 通道 0-15。且这些成员变量只能被赋值为“0”或“1”数值。“0”代表“关”状态或“低”状态，“1”代表“开”状态或“高”状态。

第五章 数据格式转换与排列规则

第一节、AD 原码 LSB 数据转换成电压值的换算方法

首先应根据设备实际位数屏蔽掉不用的高位，然后依其所选量程，按照下表公式进行换算即可。这里只以缓冲区 ADBuffer[]中的第 1 个点 ADBuffer[0]为例。

量程(mV)	计算机语言换算公式(ANSI C 语法)	Volt 取值范围（mV）
±10000mV	Volt = (20000.00/16384)*((ADBuffer[0]^0x2000)&0x3FFF)–10000.00	[-10000, +9998.77]
±5000mV	Volt = (10000.00/16384)*((ADBuffer[0]^0x2000)&0x3FFF)–5000.00	[-5000, +4999.38]
0～5000mV	Volt = (50000.00/16384)*((ADBuffer[0])&0x3FFF)	[0, +4999.69]
0～2500mV	Volt = (2500.00/16384)*((ADBuffer[0])&0x3FFF)	[0, +2499.84]

下面举例说明各种语言的换算过程（以±10000mV 量程为例）

Visual C++&C++Builder：

Lsb = (ADBuffer[0] ^ 0x2000) & 0x3FFF;
Volt = (20000.00/16384) * Lsb -10000.00;

Visual Basic：

Lsb = ((ADBuffer [0]) Xor &H2000) And &0H3FFF
Volt = (20000.00/16384) * Lsb – 10000.00

Delphi:

Lsb： = ((ADBuffer[0]) XOR $2000) And $3FFF;
Volt：= (200000.0/16384) * Lsb – 10000.00;

LabVIEW:

请参考相关演示程序。

第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

单通道采集，假如此时选择通道 5，其排放规则如下：

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
通道号	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	…

两通道采集(假如通道 0 和 1)：

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
通道号	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	…

四通道采集(假如通道 0~通道 3)：

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
通道号	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	…

其他通道方式以此类推。

如果用户是进行连续不间断循环采集，即用户只进行一次初始化设备操作，然后不停的从设备上读取 AD 数据，那么需要用户特别注意的是应处理好各通道数据排列和对齐的问题，尤其是在任意通道数采集时。否则，用户无法将规则排放在缓冲区中的各通道数据正确分离出来。那怎样正确处理呢？我们建议的方法是，每次从设备上读取的点数应置为所选通道数量的整数倍长，这样便能保证每读取的这批数据在缓冲区中的相应位置始终固定对应于某一个通道的数据。比如用户要求对 1、2 两个 AD 通道的数据进行连续循环采集，则置每次读取长度为其 2 的整倍长 2n(n 为每个通道的点数)，这里设为 2048。试想，如此一来，每次读取的 2048 个点中的第一个点始终对应于 1 通道数据，第二个点始终对应于 2 通道，第三个点再应于 1 通道，第四个点再对应于 2 通道……以此类推。直到第 2047 个点对应于 1 通道数据，第 2048 个点对应 2 通道。这样一来，每次读取的段长正好包含了从首通道到末通道的完整轮回，如此一来，用户只须按通道排列规则，按正常的处理方法循环处理每一批数据。而对于其他情况也是如此，比如 3 个通道采集，则可以使用 3n(n 为每个通道的点数)的长度采集。为了更加详细地说明问题，请参考下表（演示的是采集 1、2、3 共三个通道的情况）。由于使用连续采样方式，所以表中的数据序列一行的数字变化说明了数据采样的连续性，即随着时间的延续，数据的点数连续递增，直至用户停止设备为止，从而形成了一个有相当长度的连续不间断的多通道数据链。而通道序列一行则说明了随着连续采样的延续，其各通道数据在其整个数据链中的排放次序，这是一种非常规则而又绝对严格的顺序。但是这个相当长度的多通道数据链则不可能一次通过设备对象函数如 ReadDeviceProAD_X 函数读回，即便不考虑是否能一次读完的问题，仅对于用户的实时数据处理要求来说，一次性读取那么长的数据，则往往是相当矛盾的。因此我们就得分若干次分段读取。但怎样保证既方便处理，又不易出错，而且还高效呢？还是

正如前面所说，采用通道数的整数倍长读取每一段数据。如表中列举的方法 1（为了说明问题，我们每读取一段数据只读取 2n 即 3*2＝6 个数据）。从方法 1 不难看出，每一段缓冲区中的数据在相同缓冲区索引位置都对应于同一个通道。而在方法 2 中由于每次读取的不是通道整数倍长，则出现问题，从表中可以看出，第一段缓冲区中的 0 索引位置上的数据对应的是第 1 通道，而第二段缓冲区中的 0 索引位置上的数据则对应于第 2 通道的数据，而第三段缓冲区中的数据则对应于第 3 通道……，这显然不利于循环有效处理数据。

在实际应用中，我们在遵循以上原则时，应尽可能地使每一段缓冲足够大，这样，可以一定程度上减少数据采集程序和数据处理程序的 CPU 开销量。

数据序列	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	…
通道序列	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	…
方法 1	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	…
缓冲区号	第一段缓冲						第二段缓冲区						第三段缓冲区						第 n 段缓冲			
方法 2	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	…
	第一段缓冲区				第二段缓冲区				第三段缓冲区				第四段缓冲区				第五段缓冲区				第 n 段缓	

第三节、AD 测试应用程序创建并形成的数据文件格式

首先该数据文件从始端 0 字节位置开始往后至第 HeadSizeBytes 字节位置宽度属于文件头信息，而从 HeadSizeBytes 开始才是真正的 AD 数据。HeadSizeBytes 的取值通常等于本头信息的字节数大小。文件头信息包含的内容如下结构体所示。对于更详细的内容请参考 Visual C++高级演示工程中的 UserDef.h 文件。

```
typedef struct _FILE_HEADER
{
    LONG BusType;               // 1:PCI, 2:USB, 3:ISA, 4:PC104
    LONG DeviceID;              // 0x2006
    LONG HeaderSizeBytes;       //文件头信息长度

    LONG VoltBottomRange;       // 量程下限(mV)
    LONG VoltTopRange;          // 量程上限(mV)

    LONG ChannelCount;
    CHANNEL_ARRAY_AD ChannelArray[32]; // 采样阵列(包括通道和增益)
    LONG Frequency;
    LONG FileEndFlag;
} FILE_HEADER, *PFILE_HEADER;
```

AD 数据的格式为 16 位二进制格式，它的排放规则与在 ADBuffer 缓冲区排放的规则一样，即每 16 位二进制(字)数据对应一个 16 位 AD 数据。您只需要先开辟一个 16 位整型数组或缓冲区，然后将磁盘数据从指定位置(即双字节对齐的某个位置)读入数组或缓冲区，然后访问数组中的每个元素，即是对相应 AD 数据的访问。

第四节、DA 电压值转换成 LSB 原码数据的换算方法

量程(伏)	计算机语言换算公式(标准 C 语法)	Lsb 取值范围
0～5000mV	Lsb = Volt / (5000.00 / 4096)	[0，4095]
0～10000mV	Lsb = Vol t/ (10000.00 / 4096)	[0，4095]
±5000mV	Lsb = Volt / (10000.00 / 4096) +2048	[0，4095]
±10000mV	Lsb = Volt / (20000.00 / 4096) + 2048	[0，4095]

第六章 上层用户函数接口应用实例

第一节、怎样使用ReadDeviceProAD_NotEmpty函数直接取得AD数据

Visual C++ & C++Builder:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]╯[阿尔泰测控演示系统]╯[PCI2006 连续分组 AD 采样卡]╯[Microsoft Visual C++]╯[简易代码演示]╯[AD 非空方式]

第二节、怎样使用ReadDeviceProAD_Half函数直接取得AD数据

Visual C++ & C++Builder:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]╯[阿尔泰测控演示系统]╯[PCI2006 连续分组 AD 采样卡]╯[Microsoft Visual C++]╯[简易代码演示]╯[AD 半满方式]

第三节、怎样使用中断方式取得 AD 数据

Visual C++ & C++Builder:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]╯[阿尔泰测控演示系统]╯[PCI2006 连续分组 AD 采样卡]╯[Microsoft Visual C++]╯[简易代码演示]╯[AD 中断方式]

第四节、怎样使用WriteDeviceDA函数取得DA数据

Visual C++ & C++Builder:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]╯[阿尔泰测控演示系统]╯[PCI2006 连续分组 AD 采样卡]╯[Microsoft Visual C++]╯[简易代码演示]╯[DA 输出]

第五节、怎样使用GetDeviceDI函数进行更便捷的数字开关量输入操作

Visual C++ & C++Builder:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]╯[阿尔泰测控演示系统]╯[PCI2006 连续分组 AD 采样卡]╯[Microsoft Visual C++]╯[简易代码演示]╯[DIO…]

第六节、怎样使用SetDeviceDO函数进行更便捷的数字开关量输出操作

Visual C++ & C++Builder:

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序]╯[阿尔泰测控演示系统]╯[PCI2006 连续分组 AD 采样卡]╯[Microsoft Visual C++]╯[简易代码演示]╯[DIO…]

第七章 高速大容量、连续不间断数据采集及存盘技术详解

与ISA、USB设备同理，使用子线程跟踪AD转换进度，并进行数据采集是保持数据连续不间断的最佳方案。但是与ISA总线设备不同的是，PCI设备在这里不使用动态指针去同步AD转换进度，因为ISA设备环形内存池的动态指针操作是一种软件化的同步，而PCI设备不再有软件化的同步，而完全由硬件和驱动程序自动完成。这样一来，用户要用程序方式实现连续数据采集，其软件实现就显得极为容易。每次用ReadDeviceProAD_X函数

读取AD数据时，那么设备驱动程序会按照AD转换进度将AD数据一一放进用户数据缓冲区，当完成该次所指定的点数时，它便会返回，当您再次用这个函数读取数据时，它会接着上一次的位置传递数据到用户数据缓冲区。只是要求每两次ReadDeviceProAD_NotEmpty(或者ReadDeviceProAD_Half)之间的时间间隔越短越好。

但是由于我们的设备是通常工作在一个单 CPU 多任务的环境中，由于任务之间的调度切换非常平凡，特别是当用户移动窗口、或弹出对话框等，则会使当前线程猛地花掉大量的时间去处理这些图形操作，因此如果处理不当，则将无法实现高速连续不间断采集，那么如何更好的克服这些问题呢？用子线程则是必须的（在这里我们称之为数据采集线程），但这还不够，必须要求这个线程是绝对的工作者线程，即这个线程在正常采集中不能有任何窗口等图形操作。只有这样，当用户进行任何窗口操作时，这个线程才不会被堵塞，因此可以保证其正常连续的数据采集。但是用户可能要问，不能进行任何窗口操作，那么我如何将采集的数据显示在屏幕上呢？其实很简单，再开辟一个子线程，我们称之数据处理线程，也叫用户界面线程。最初，数据处理线程不做任何工作，而是在 Win32 API 函数 WaitForSingleObject 的作用下进入睡眠状态，此时它基本不消耗 CPU 时间，即可保证其他线程代码有充分的运行机会（这里当然主要指数据采集线程），当数据采集线程取得指定长度的数据到用户空间时，则再用 Win32 API 函数 SetEvent 将指定事件消息发送给数据处理线程，则数据处理线程即刻恢复运行状态，迅速对这批数据进行处理，如计算、在窗口绘制波形、存盘等操作。

可能用户还要问，既然数据处理线程是非工作者线程，那么如果用户移动窗口等操作堵塞了该线程，而数据采集线程则在不停地采集数据，那数据处理线程难道不会因此而丢失采集线程发来的某一段数据吗？如果不另加处理，这个情况肯定有发生的可能。但是，我们采用了一级缓冲队列和二级缓冲队列的设计方案，足以避免这个问题。即假设数据采集线程每一次从设备上取出 8K数据，那么我们就创建一个缓冲队列，在用户程序中最简单的办法就是开辟一个两维数组如ADBuffer [SegmentCount][SegmentSize]， 我们将SegmentSize视为数据采集线程每次采集的数据长度，SegmentCount则为缓冲队列的成员个数。您应根据您的计算机物理内存大小和总体使用情况来设定这个数。假如我们设成 32，则这个缓冲队列实际上就是数组ADBuffer [32][8192]的形式。那么如何使用这个缓冲队列呢？方法很简单，它跟一个普通的缓冲区如一维数组差不多，唯一不同是，两个线程首先要通过改变SegmentCount字段的值，即这个下标Index的值来填充和引用由Index下标指向某一段SegmentSize长度的数据缓冲区。需要注意的是两个线程不共用一个Index下标变量。具体情况是当数据采集线程在AD部件被InitDeviceProAD初始化之后，首次采集数据时，则将自己的ReadIndex下标置为 0，即用第一个缓冲区采集AD数据。当采集完后，则向数据处理线程发送消息，且两个线程的公共变量SegmentCount加 1，（注意SegmentCount变量是用于记录当前时刻缓冲队列中有多少个已被数据采集线程使用了，但是却没被数据处理线程处理掉的缓冲区数量。）然后再接着将ReadIndex偏移至 1，再用第二个缓冲区采集数据。再将SegmentCount加 1，直到ReadIndex等于 31 为止，然后再回到 0 位置，重新开始。而数据处理线程则在每次接受到消息时判断有多少由于自已被堵塞而没有被处理的缓冲区个数，然后逐一进行处理，最后再从SegmentCount变量中减去在所接受到的当前事件下所处理的缓冲区个数，具体处理哪个缓冲区由CurrentIndex指向。因此，即便应用程序突然很忙，使数据处理线程没有时间处理已到来的数据，但是由于缓冲区队列的缓冲作用，可以让数据采集线程先将数据连续缓存在这个区域中，由于这个缓冲区可以设计得比较大，因此可以缓冲很大的时间，这样即便是数据处理线程由于系统的偶而繁忙而被堵塞，也很难使数据丢失。而且通过这种方案，用户还可以在数据采集线程中对SegmentCount加以判断，观察其值是否大于了 32，如果大于，则缓冲区队列肯定因数据处理采集的过度繁忙而被溢出，如果溢出即可报警。因此具有强大的容错处理。

图 7.1 便形象的演示了缓冲队列处理的方法。可以看出，最初设备启动时，数据采集线程在往 ADBuffer[0]里面填充数据时，数据处理线程便在 WaitForSingleObject 的作用下睡眠等待有效数据。当 ADBuffer[0]被数据采集线程填充满后，立即给数据处理线程 SetEvent 发送通知 hEvent，便紧接着开始填充 ADBuffer[1]，数据处理线程接到事件后，便醒来开始处理数据 ADBuffer[0]缓冲。它们就这样始终差一个节拍。如虚线箭头所示。

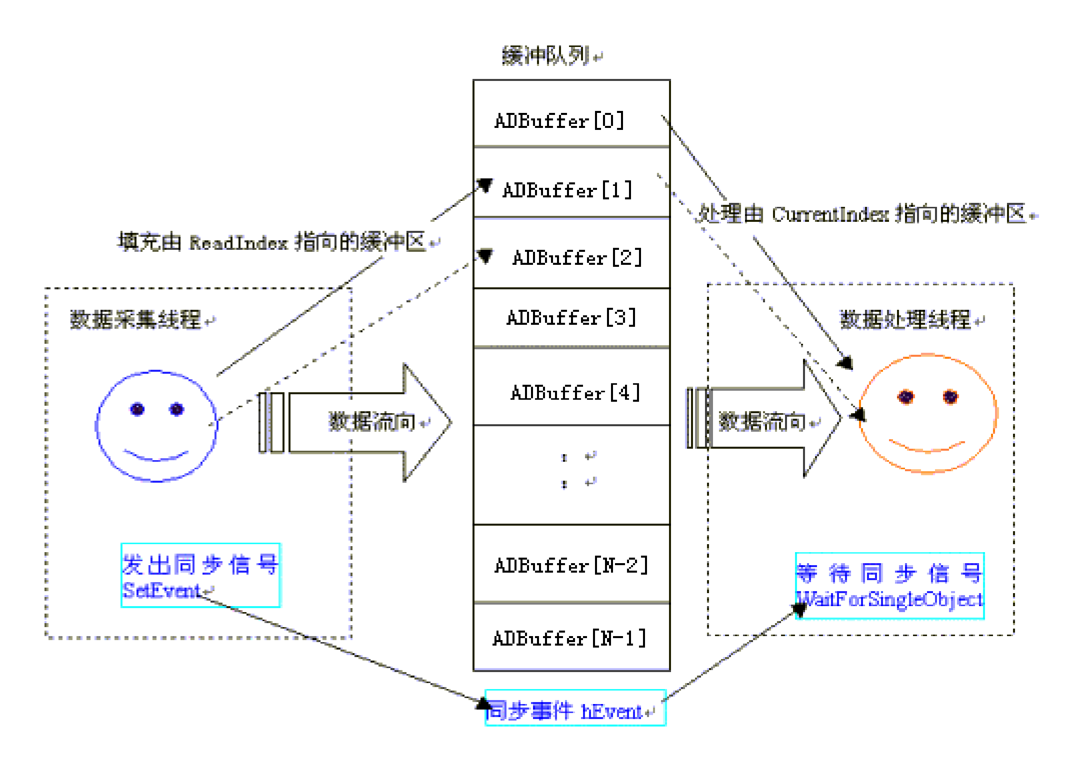


图 7.1

第一节、使用程序查询方式实现该功能

下面用 **Visual C++**程序举例说明。

一、使用[ReadDeviceProAD_NotEmpty]函数读取设备上的AD数据（它使用FIFO的非空标志）

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(ADDoc.h 和 ADDoc.cpp， ADThread.h 和 ADThread.cpp)。

[程序]╛[阿尔泰测控演示系统]╛[PCI2006 连续分组 AD 采样卡]╛[Microsoft Visual C++]╛[高级代码演示]╛[演示源程序]

然后，您着重参考 ADDoc.cpp 源文件中以下函数：

void CADDoc::StartDeviceAD() // 启动线程函数
BOOL MyStartDeviceAD(HANDLE hDevice); // 位于 ADThread.cpp
UINT ReadDataThread_Npt (PVOID pThreadPara) // 读数据线程，位于 ADThread.cpp
UINT ProcessDataThread(PVOID pThreadPara) // 绘制数据线程
BOOL MyStopDeviceAD(HANDLE hDevice); // 位于 ADThread.cpp
void CADDoc::StopDeviceAD() // 终止采集函数

二、使用[ReadDeviceProAD_Half]函数读取设备上的AD数据（它使用FIFO的半满标志）

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(ADDoc.h 和 ADDoc.cpp， ADThread.h 和 ADThread.cpp)。

[程序]╛[阿尔泰测控演示系统]╛[PCI2006 连续分组 AD 采样卡]╛[Microsoft Visual C++]╛[高级代码演示]╛[演示源程序]

然后，您着重参考 ADDoc.cpp 源文件中以下函数：

void CADDoc::StartDeviceAD() // 启动线程函数
BOOL MyStartDeviceAD(HANDLE hDevice); // 位于 ADThread.cpp
UINT ReadDataThread_Half (PVOID pThreadPara) // 读数据线程，位于 ADThread.cpp
UINT ProcessDataThread(PVOID pThreadPara) // 绘制数据线程
BOOL MyStopDeviceAD(HANDLE hDevice); // 位于 ADThread.cpp
void CADDoc::StopDeviceAD() // 终止采集函数

当然用 FIFO 非空标志读取 AD 数据，能获得接近 FIFO 总容量的栈深度，这样用户在两批数据之间，便有更多的时间来处理某些数据。而用半满标志，则最多只能达到 FIFO 总容量的二分之一的栈深度，那么用户

在两批数据之间处理数据的时间会相对短些，但是半满读取时，查询 AD 转换标志的时间则最少。当然究意那种方案最好，还得看用户的实际需要。

第二节、使用中断方式实现该功能

其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(ADDoc.cpp 和 ADThread.cpp)。

[程序]╛**[阿尔泰测控演示系统]**╛ **[PCI2006 连续分组 AD 采样卡]**╛ **[Microsoft Visual C++]**╛ **[高级代码演示]**╛ **[演示源程序]**

然后，您着重参考 ADDoc.cpp 源文件中以下函数：

void CADDoc:: OnStartDeviceAD () // 采集线程和处理线程的启动函数

BOOL StartDeviceAD_Int () // 启动采集线程函数

UINT ReadDataThread_Int() // 采集线程函数

BOOL StopDeviceAD_Int() // 采集线程的终止函数

UINT DrawWindowProc () // 绘制数据线程

void CADDoc:: OnStopDeviceAD () // 终止采集函数

第八章 共用函数介绍

这部分函数不参与本设备的实际操作，它只是为您编写数据采集与处理程序时的有力手段，使您编写应用程序更容易，使您的应用程序更高效。

第一节、公用接口函数总列表（每个函数省略了前缀“PCI2006_”）

函数名	函数功能	备注
① **PCI 总线内存映射寄存器操作函数**		
GetDeviceAddr	取得指定 PCI 设备寄存器操作基地址	底层用户
WriteRegisterByte	以字节(8Bit)方式写寄存器端口	底层用户
WriteRegisterWord	以字(16Bit)方式写寄存器端口	底层用户
WriteRegisterULong	以双字(32Bit)方式写寄存器端口	底层用户
ReadRegisterByte	以字节(8Bit)方式读寄存器端口	底层用户
ReadRegisterWord	以字(16Bit)方式读寄存器端口	底层用户
ReadRegisterULong	以双字(32Bit)方式读寄存器端口	底层用户
② **ISA 总线 I/O 端口操作函数**		
WritePortByte	以字节(8Bit)方式写 I/O 端口	用户程序操作端口
WritePortWord	以字(16Bit)方式写 I/O 端口	用户程序操作端口
WritePortULong	以无符号双字(32Bit)方式写 I/O 端口	用户程序操作端口
ReadPortByte	以字节(8Bit)方式读 I/O 端口	用户程序操作端口
ReadPortWord	以字(16Bit)方式读 I/O 端口	用户程序操作端口
ReadPortULong	以无符号双字(32Bit)方式读 I/O 端口	用户程序操作端口
③ **创建 Visual Basic 子线程，线程数量可达 32 个以上**		
CreateVBThread	在 VB 环境中建立子线程对象	
TerminateVBThread	终止 VB 的子线程	
CreateSystemEvent	创建系统内核事件对象	用于线程同步或中断
ReleaseSystemEvent	释放系统内核事件对象	
DelayTimeUs	高效高精度延时函数	不消耗 CPU 时间
④ **文件对象操作函数**		
CreateFileObject	初始设备文件对象	
WriteFile	请求文件对象写用户数据到磁盘文件	
ReadFile	请求文件对象读数据到用户空间	
SetFileOffset	设置文件指针偏移	
GetFileLength	取得文件长度	

ReleaseFile	释放已有的文件对象	
GetDiskFreeBytes	取得指定磁盘的可用空间(字节)	适用于所有设备

第二节、PCI 内存映射寄存器操作函数原型说明

◆ 取得指定内存映射寄存器的线性地址和物理地址

函数原型：

Visual C++ & C++ Builder:

```
BOOL GetDeviceAddr( HANDLE hDevice,
                    PULONG LinearAddr,
                    PULONG PhysAddr,
                    int RegisterID = 0)
```

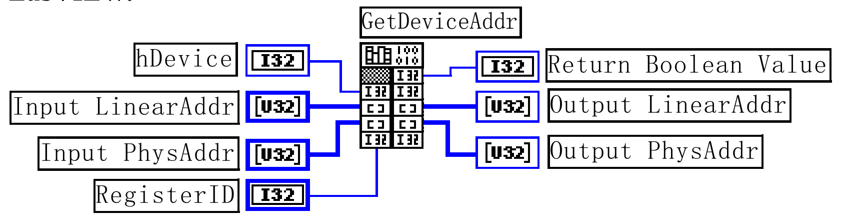
Visual Basic:

```
Declare Function GetDeviceAddr Lib "PCI2006" (ByVal hDevice As Long, _
                                    ByRef LinearAddr As Long, _
                                    ByRef PhysAddr As Long, _
                                    ByVal RegisterID As Integer = 0) As Boolean
```

Delphi:

```
Function GetDeviceAddr(hDevice : Integer;
                    LinearAddr : Pointer;
                    PhysAddr : Pointer;
                    RegisterID : Integer = 0) : Boolean;
                    StdCall; External 'PCI2006' Name ' GetDeviceAddr ';
```

LabVIEW:



功能：取得 PCI 设备指定的内存映射寄存器的线性地址。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

LinearAddr 指针参数，用于取得的映射寄存器指向的线性地址，RegisterID 指定的寄存器组属于 MEM 模式时该值不应为零，也就是说它可用于 WriteRegisterX 或 ReadRegisterX（X 代表 Byte、ULong、Word）等函数，以便于访问设备寄存器。它指明该设备位于系统空间的虚拟位置。但如果 RegisterID 指定的寄存器组属于 I/O 模式时该值通常为零，您不能通过以上函数访问设备。

PhysAddr 指针参数，用于取得的映射寄存器指向的物理地址，它指明该设备位于系统空间的物理位置。如果由 RegisterID 指定的寄存器组属于 I/O 模式，则可用于 WritePortX 或 ReadPortX（X 代表 Byte、ULong、Word）等函数，以便于访问设备寄存器。

RegisterID 指定映射寄存器的 ID 号，其取值范围为[0，5]，通常情况下，用户应使用 0 号映射寄存器，特殊情况下，我们为用户加以申明。本设备的寄存器组 ID 定义如下：

常量名	常量值	功能定义
PCI2006_REG_MEM_PLXCHIP	0x0000	0 号寄存器对应 PLX 芯片所使用的内存模式基地址(使用 LinearAddr)
PCI2006_REG_IO_PLXCHIP	0x0001	1 号寄存器对应 PLX 芯片所使用的 IO 模式基地址(使用 PhysAddr)
PCI2006_REG_IO_CPLD	0x0002	2 号寄存器对应板上控制单元所使用的 IO 模式基地址(使用 PhysAddr)
PCI2006_REG_IO_ADFIFO	0x0003	3 号寄存器对应板上 AD FIFO 缓冲区所使用的 IO 模式基地址(使用 PhysAddr)

返回值：如果执行成功，则返回TRUE，它表明由RegisterID指定的映射寄存器的无符号 32 位线性地址和物理地址被正确返回，否则会返回FALSE，同时还要检查其LinearAddr和PhysAddr是否为 0，若为 0 则依然视为失败。用户可用GetLastErrorEx捕获当前错误码，并加以分析。

相关函数： CreateDevice GetDeviceAddr WriteRegisterByte

WriteRegisterWord	WriteRegisterULong	ReadRegisterByte
ReadRegisterWord	ReadRegisterULong	ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr;
hDevice = CreateDevice(0);
if(!GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0))
{
    AfxMessageBox(“取得设备地址失败…”);
}
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr   As Long
hDevice = CreateDevice(0)
if Not GetDeviceAddr(hDevice, LinearAddr,   PhysAddr, 0) then
    MsgBox “取得设备地址失败…”
End If
    :
```

◆ 以单字节（即 8 位）方式写 **PCI** 内存映射寄存器的某个单元

函数原型：

Visual C++ & C++ Builder:

```
BOOL WriteRegisterByte( HANDLE hDevice,
                        ULONG LinearAddr,
                        ULONG OffsetBytes,
                        BYTE Value)
```

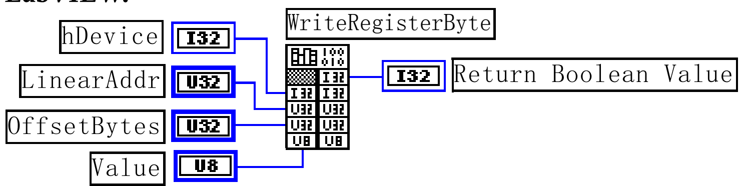
Visual Basic:

```
Declare Function WriteRegisterByte Lib "PCI2006" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long, _
                                                  ByVal Value As Byte ) As Boolean
```

Delphi:

```
Function WriteRegisterByte(  hDevice : Integer;
                             LinearAddr : LongWord;
                             OffsetBytes : LongWord;
                             Value : Byte) : Boolean;
                             StdCall; External 'PCI2006' Name ' WriteRegisterByte ';
```

LabVIEW:



功能：以单字节（即 8 位）方式写 PCI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

LinearAddr PCI设备内存映射寄存器的线性基地址，它的值应由GetDeviceAddr确定。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 WriteRegisterByte函数所访问的映射寄存器的内存单元。

Value 输出 8 位整数。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数：

CreateDevice	GetDeviceAddr	WriteRegisterByte
WriteRegisterWord	WriteRegisterULong	ReadRegisterByte
ReadRegisterWord	ReadRegisterULong	ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
        :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,   &PhysAddr, 0) )
{
        AfxMessageBox "取得设备地址失败…";
}
OffsetBytes = 100;      // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterByte(hDevice, LinearAddr, OffsetBytes, 0x20);   // 往指定映射寄存器单元写入 8 位的十六进制数据 20
ReleaseDevice( hDevice );   // 释放设备对象
        :
```

Visual Basic 程序举例:

```
        :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterByte( hDevice, LinearAddr, OffsetBytes, &H20)
ReleaseDevice(hDevice)
        :
```

♦ **以双字节（即 16 位）方式写 PCI 内存映射寄存器的某个单元**

函数原型：

Visual C++ & C++ Builder:

```
BOOL WriteRegisterWord( HANDLE hDevice,
                        ULONG LinearAddr,
                        ULONG OffsetBytes,
                        WORD Value)
```

Visual Basic:

```
Declare Function WriteRegisterWord Lib "PCI2006" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long, _
                                                  ByVal Value As Integer) As Boolean
```

Delphi:

```
Function WriteRegisterWord( hDevice : Integer;
                            LinearAddr : LongWord;
                            OffsetBytes : LongWord;
                            Value : Word) : Boolean;
                            StdCall; External 'PCI2006' Name ' WriteRegisterWord ';
```

LabVIEW:

功能：以双字节（即 16 位）方式写 PCI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

LinearAddr PCI设备内存映射寄存器的线性基地址，它的值应由GetDeviceAddr确定。

OffsetBytes 相对于LinearAddr线性基地址的偏移字节数，它与LinearAddr两个参数共同确定WriteRegisterWord函数所访问的映射寄存器的内存单元。

Value 输出 16 位整型值。

返回值：无。

相关函数： CreateDevice　GetDeviceAddr　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte

ReadRegisterWord　　ReadRegisterULong　　ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
        :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,   &PhysAddr, 0) )
{
        AfxMessageBox "取得设备地址失败…";
}
OffsetBytes = 100;      // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterWord(hDevice, LinearAddr, OffsetBytes, 0x2000);   // 往指定映射寄存器单元写入 16 位的十六进制数据
ReleaseDevice( hDevice );   // 释放设备对象
        :
```

Visual Basic 程序举例:

```
        :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes=100
WriteRegisterWord( hDevice, LinearAddr, OffsetBytes, &H2000)
ReleaseDevice(hDevice)
        :
```

♦ 以四字节（即 32 位）方式写 PCI 内存映射寄存器的某个单元

函数原型：

Visual C++ & C++ Builder:

```
BOOL WriteRegisterULong( HANDLE hDevice,
                         ULONG LinearAddr,
                         ULONG OffsetBytes,
                         ULONG Value)
```

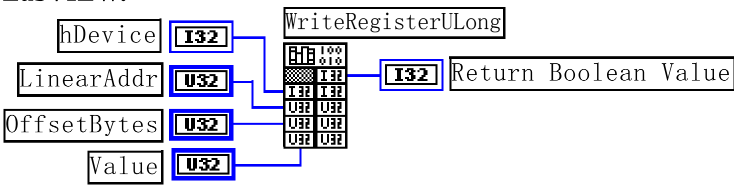
Visual Basic:

```
Declare Function WriteRegisterULong Lib "PCI2006" (ByVal hDevice As Long, _
                                                   ByVal LinearAddr As Long, _
                                                   ByVal OffsetBytes As Long, _
                                                   ByVal Value As Long) As Boolean
```

Delphi:

```
Function WriteRegisterULong(hDevice : Integer;
                            LinearAddr : LongWord;
                            OffsetBytes : LongWord;
                            Value : LongWord) : Boolean;
                            StdCall; External 'PCI2006' Name ' WriteRegisterULong ';
```

LabVIEW:



功能：以四字节（即 32 位）方式写 PCI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

LinearAddr PCI设备内存映射寄存器的线性基地址，它的值应由GetDeviceAddr确定。

OffsetBytes 相对于LinearAddr线性基地址的偏移字节数，它与LinearAddr两个参数共同确定WriteRegisterULong函数所访问的映射寄存器的内存单元。

Value 输出 32 位整型值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数：　CreateDevice　　GetDeviceAddr　　WriteRegisterByte

WriteRegisterWord　　WriteRegisterULong　　ReadRegisterByte
ReadRegisterWord　　ReadRegisterULong　　ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
        :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,   &PhysAddr, 0) )
{
        AfxMessageBox "取得设备地址失败…";
}
OffsetBytes=100;// 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterULong(hDevice, LinearAddr, OffsetBytes, 0x20000000);   // 往指定映射寄存器单元写入 32 位的十六进制数据
ReleaseDevice( hDevice );   // 释放设备对象
        :
```

Visual Basic 程序举例:

```
        :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterULong( hDevice, LinearAddr, OffsetBytes, &H20000000)
ReleaseDevice(hDevice)
        :
```

- 以单字节（即 8 位）方式读 **PCI** 内存映射寄存器的某个单元

函数原型：

Visual C++ & C++ Builder:

```
BYTE ReadRegisterByte( HANDLE hDevice,
                       ULONG LinearAddr,
                       ULONG OffsetBytes)
```

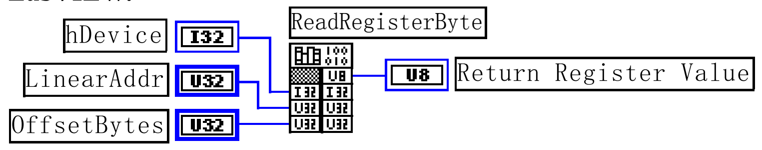
Visual Basic:

```
Declare Function ReadRegisterByte Lib "PCI2006" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long) As Byte
```

Delphi:

```
Function ReadRegisterByte(hDevice : Integer;
                          LinearAddr : LongWord;
                          OffsetBytes : LongWord) : Byte;
                          StdCall; External 'PCI2006' Name ' ReadRegisterByte ';
```

LabVIEW:



功能：以单字节（即 8 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

LinearAddr　PCI设备内存映射寄存器的线性基地址，它的值应由GetDeviceAddr确定。

OffsetBytes　相对于LinearAddr线性基地址的偏移字节数，它与LinearAddr两个参数共同确定ReadRegisterByte函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 8 位数据。

相关函数：　CreateDevice　　GetDeviceAddr　　WriteRegisterByte
WriteRegisterWord　　WriteRegisterULong　　ReadRegisterByte
ReadRegisterWord　　ReadRegisterULong　　ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr,   PhysAddr, OffsetBytes;
BYTE   Value;
hDevice = CreateDevice(0);  // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);    // 取得 PCI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;     // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterByte(hDevice, LinearAddr, OffsetBytes);  // 从指定映射寄存器单元读入 8 位数据
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Byte
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterByte( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
     :
```

- **以双字节（即 16 位）方式读 PCI 内存映射寄存器的某个单元**

函数原型：

Visual C++ & C++ Builder:

```
WORD ReadRegisterWord( HANDLE hDevice,
                       ULONG LinearAddr,
                       ULONG OffsetBytes)
```

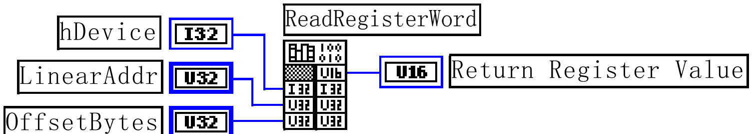
Visual Basic:

```
Declare Function ReadRegisterWord Lib "PCI2006" (  ByVal hDevice As Long, _
                                                   ByVal LinearAddr As Long, _
                                                   ByVal OffsetBytes As Long) As Integer
```

Delphi:

```
Function ReadRegisteWord(hDevice : Integer;
                         LinearAddr : LongWord;
                         OffsetBytes : LongWord) : Word;
                         StdCall; External 'PCI2006' Name ' ReadRegisterWord ';
```

LabVIEW:



功能：以双字节（即 16 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

LinearAddr PCI设备内存映射寄存器的线性基地址，它的值应由GetDeviceAddr确定。

OffsetBytes 相对于LinearAddr线性基地址的偏移字节数，它与LinearAddr两个参数共同确定ReadRegisterWord函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 16 位数据。

相关函数:	CreateDevice	GetDeviceAddr	WriteRegisterByte
	WriteRegisterWord	WriteRegisterULong	ReadRegisterByte
	ReadRegisterWord	ReadRegisterULong	ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr,   PhysAddr, OffsetBytes;
WORD Value;
hDevice = CreateDevice(0);  // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);    // 取得 PCI 设备 0 号映射寄存器的线性基地址
```

```
OffsetBytes = 100;    // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterWord(hDevice, LinearAddr, OffsetBytes);  // 从指定映射寄存器单元读入 16 位数据
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Word
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterWord( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
    :
```

- ◆ 以四字节（即 **32** 位）方式读 **PCI** 内存映射寄存器的某个单元

函数原型：

Visual C++ & C++ Builder:

```
ULONG ReadRegisterULong( HANDLE hDevice,
                         ULONG LinearAddr,
                         ULONG OffsetBytes)
```

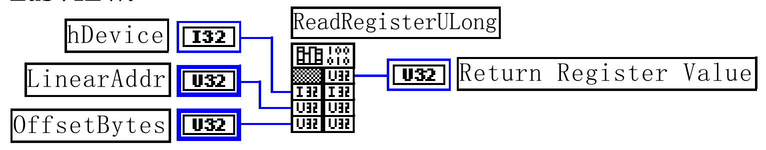
Visual Basic:

```
Declare Function ReadRegisterULong Lib "PCI2006" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long) As Long
```

Delphi:

```
Function ReadRegisterULong(hDevice : Integer;
                           LinearAddr : LongWord;
                           OffsetBytes : LongWord) : LongWord;
                           StdCall; External 'PCI2006' Name ' ReadRegisterULong ';
```

LabVIEW:



功能：以四字节（即 32 位）方式读 PCI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

LinearAddr PCI设备内存映射寄存器的线性基地址，它的值应由GetDeviceAddr确定。

OffsetBytes 相对与LinearAddr线性基地址的偏移字节数，它与LinearAddr两个参数共同确定WriteRegisterULong函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 32 位数据。

相关函数：

CreateDevice	GetDeviceAddr	WriteRegisterByte
WriteRegisterWord	WriteRegisterULong	ReadRegisterByte
ReadRegisterWord	ReadRegisterULong	ReleaseDevice

Visual C++ & C++ Builder 程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr,  PhysAddr, OffsetBytes;
ULONG  Value;
hDevice = CreateDevice(0);  // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);   // 取得 PCI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;    // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterULong(hDevice, LinearAddr, OffsetBytes);  // 从指定映射寄存器单元读入 32 位数据
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
        :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterULong( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
        :
```

第三节、IO 端口读写函数原型说明

注意：若您想在 **WIN2K** 系统的 **User** 模式中直接访问 **I/O** 端口，那么您可以安装光盘中 **ISA\CommUser** 目录下的公用驱动，然后调用其中的 **WritePortByteEx** 或 **ReadPortByteEx** 等有“**Ex**”后缀的函数即可。

◆ 以单字节**(8Bit)**方式写 **I/O** 端口

函数原型：

Visual C++ & C++ Builder:

```
BOOL WritePortByte (HANDLE hDevice,
                    UINT nPort,
                    BYTE Value)
```

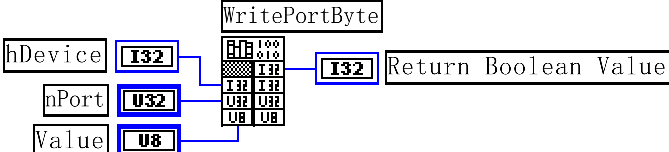
Visual Basic:

```
Declare Function WritePortByte Lib "PCI2006" ( ByVal hDevice As Long, _
                                               ByVal nPort As Long, _
                                               ByVal Value As Byte) As Boolean
```

Delphi:

```
Function WritePortByte(hDevice : Integer;
                       nPort : LongWord;
                       Value : Byte) : Boolean;
                       StdCall; External 'PCI2006' Name ' WritePortByte ';
```

LabVIEW:



功能：以单字节(8Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

nPort 设备的 I/O 端口号。

Value 写入由 nPort 指定端口的值。

返回值：若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。

相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

◆ 以双字**(16Bit)**方式写 **I/O** 端口

函数原型：

Visual C++ & C++ Builder:

```
BOOL WritePortWord (HANDLE hDevice,
                    UINT nPort,
                    WORD Value)
```

Visual Basic:

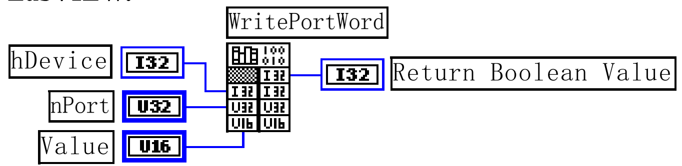
```
Declare Function WritePortWord Lib "PCI2006" (ByVal hDevice As Long, _
                                              ByVal nPort As Long, _
                                              ByVal Value As Integer) As Boolean
```

Delphi:

```
Function WritePortWord(hDevice : Integer;
                       nPort : LongWord;
```

Value : Word) : Boolean;
StdCall; External 'PCI2006' Name ' WritePortWord ';

LabVIEW:



功能：以双字(16Bit)方式写 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
nPort 设备的 I/O 端口号。
Value 写入由 nPort 指定端口的值。
返回值：若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。
相关函数： CreateDevice WritePortByte WritePortWord
WritePortULong ReadPortByte ReadPortWord

- **以四字节(32Bit)方式写 I/O 端口**

函数原型：
Visual C++ & C++ Builder:

```
BOOL WritePortULong(HANDLE hDevice,
                    UINT nPort,
                    ULONG Value)
```

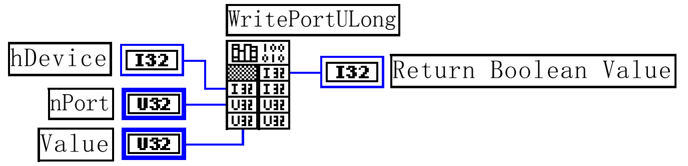
Visual Basic:

```
Declare Function WritePortULong Lib "PCI2006" (ByVal hDevice As Long, _
                                               ByVal nPort As Long, _
                                               ByVal Value As Long ) As Boolean
```

Delphi:

```
Function WritePortULong(hDevice : Integer;
                        nPort : LongWord;
                        Value : LongWord) : Boolean;
                        StdCall; External 'PCI2006' Name ' WritePortULong ';
```

LabVIEW:



功能：以四字节(32Bit)方式写 I/O 端口。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
nPort 设备的 I/O 端口号。
Value 写入由 nPort 指定端口的值。
返回值：若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。
相关函数： CreateDevice WritePortByte WritePortWord
WritePortULong ReadPortByte ReadPortWord

- **以单字节(8Bit)方式读 I/O 端口**

函数原型：
Visual C++ & C++ Builder:

```
BYTE ReadPortByte( HANDLE hDevice,
                   UINT nPort)
```

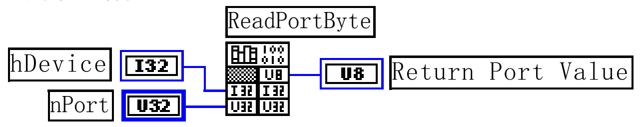
Visual Basic:

```
Declare Function ReadPortByte Lib "PCI2006" (ByVal hDevice As Long, _
                                             ByVal nPort As Long ) As Byte
```

Delphi:

Function ReadPortByte(hDevice : Integer;
nPort : LongWord) : Byte;
StdCall; External 'PCI2006' Name ' ReadPortByte ';

LabVIEW:



功能：以单字节(8Bit)方式读 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
nPort 设备的 I/O 端口号。
返回值：返回由 nPort 指定的端口的值。
相关函数： CreateDevice WritePortByte WritePortWord
WritePortULong ReadPortByte ReadPortWord

◆ **以双字节(16Bit)方式读 I/O 端口**

函数原型：

Visual C++ & C++ Builder:

WORD ReadPortWord(HANDLE hDevice,
UINT nPort)

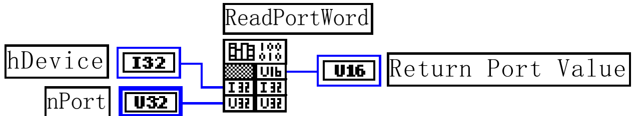
Visual Basic:

Declare Function ReadPortWord Lib "PCI2006" (ByVal hDevice As Long, _
ByVal nPort As Long) As Integer

Delphi:

Function ReadPortWord(hDevice : Integer;
nPort : LongWord) : Word;
StdCall; External 'PCI2006' Name ' ReadPortWord ';

LabVIEW:



功能：以双字节(16Bit)方式读 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
nPort 设备的 I/O 端口号。
返回值：返回由 nPort 指定的端口的值。
相关函数： CreateDevice WritePortByte WritePortWord
WritePortULong ReadPortByte ReadPortWord

◆ **以四字节(32Bit)方式读 I/O 端口**

函数原型：

Visual C++ & C++ Builder:

ULONG ReadPortULong(HANDLE hDevice,
UINT nPort)

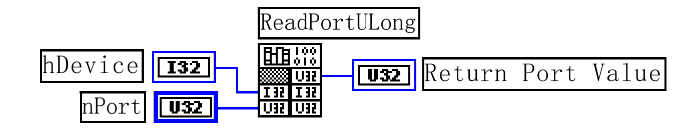
Visual Basic:

Declare Function ReadPortULong Lib "PCI2006" (ByVal hDevice As Long, _
ByVal nPort As Long) As Long

Delphi:

Function ReadPortULong(hDevice : Integer;
nPort : LongWord) : LongWord;
StdCall; External 'PCI2006' Name ' ReadPortULong ';

LabVIEW:



功能：以四字节(32Bit)方式读 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
nPort 设备的 I/O 端口号。
返回值：返回由 nPort 指定端口的值。
相关函数： CreateDevice WritePortByte WritePortWord
WritePortULong ReadPortByte ReadPortWord

第四节、线程操作函数原型说明

（如果您的 VB6.0 中线程无法正常运行，可能是 VB6.0 语言本身的问题，请选用 VB5.0）

◆ **在 VB 环境中，创建子线程对象，以实现多线程操作**

函数原型：

Visual C++ & C++ Builder:

```
BOOL CreateVBThread(HANDLE *hThread,
                    LPTHREAD_START_ROUTINE lpStartThread)
```

Visual Basic：

```
Declare Function CreateVBThread Lib "PCI2006" ( ByRef hThread As Long, _
                                     ByVal lpStartThread As Long ) As Boolean
```

功能：该函数在 VB 环境中解决了不能实现或不能很好地实现多线程的问题。通过该函数用户可以很轻松地实现多线程操作。

参数：

hThread 若成功创建子线程，该参数将返回所创建的子线程的句柄，当用户操作这个子线程时将用到这个句柄，如启动线程、暂停线程以及删除线程等。

lpStartThread作为子线程运行的函数的地址，在实际使用时，请用AddressOf关键字取得该子线程函数的地址，再传递给CreateVBThread函数。

返回值：当成功创建子线程时，返回TRUE，且所创建的子线程为挂起状态，用户需要用Win32 API函数ResumeThread函数启动它。若失败，则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。

相关函数： CreateVBThread TerminateVBThread

注意: StartThread 指向的函数或过程必须放在 VB 的模块文件中，如 PCI2006.Bas 文件中。

Visual Basic 程序举例:

```
' 在模块文件中定义子线程函数(注意参考实例)
Function NewRoutine() As Long       ' 定义子线程函数
      :                       ' 线程运行代码
NewRoutine = 1   ' 返回成功码
End Function
'
' 在窗体文件中创建子线程
     :
Dim hNewThread As Long
    If Not CreateVBThread(hNewThread, AddressOf NewRoutine) Then ' 创建新子线程
        MsgBox "创建子线程失败"
        Exit Sub
    End If
    ResumeThread (hNewThread)   '启动新线程         :
     :
```

◆ **在 VB 中，删除子线程对象**

函数原型：

Visual C++ & C++ Builder:

BOOL TerminateVBThread(HANDLE hThreadHandle)

Visual Basic:

Declare Function TerminateVBThread Lib "PCI2006" (ByVal hThreadHandle As Long) As Boolean

功能：在VB中删除由CreateVBThread创建的子线程对象。

参数：hThreadHandle 指向需要删除的子线程对象的句柄，它应由CreateVBThread创建。

返回值：当成功删除子线程对象时，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。

相关函数： CreateVBThread TerminateVBThread

Visual Basic 程序举例:

```
    :
If Not TerminateVBThread (hNewThread)   ' 终止子线程
    MsgBox "创建子线程失败"
    Exit Sub
End If
    :
```

♦ 创建内核系统事件

函数原型：

Visual C++ & C++ Builder:

HANDLE CreateSystemEvent(void)

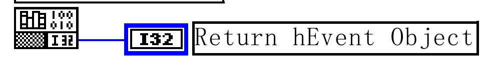
Visual Basic:

Declare Function CreateSystemEvent Lib " PCI2006 " () As Long

Delphi:

Function CreateSystemEvent() : Integer;
StdCall; External 'PCI2006' Name ' CreateSystemEvent ';

LabVIEW：



功能：创建系统内核事件对象，它将被用于中断事件响应或数据采集线程同步事件。

参数：无任何参数。

返回值：若成功，返回系统内核事件对象句柄，否则返回－1(或 INVALID_HANDLE_VALUE)。

♦ 释放内核系统事件

函数原型：

Visual C++ & C++ Builder：

BOOL ReleaseSystemEvent(HANDLE hEvent)

Visual Basic:

Declare Function ReleaseSystemEvent Lib " PCI2006 " (ByVal hEvent As Long) As Boolean

Delphi:

Function ReleaseSystemEvent(hEvent : Integer) : Boolean;
StdCall; External 'PCI2006' Name ' ReleaseSystemEvent ';

LabVIEW：

请参见相关演示程序。

功能：释放系统内核事件对象。

参数：hEvent 被释放的内核事件对象。它应由CreateSystemEvent成功创建的对象。

返回值：若成功，则返回 TRUE。

♦ 高效高精度延时

函数原型：

Visual C++ & C++ Builder：

BOOL DelayTime (HANDLE hDevice,
LONG nTime)

Visual Basic：

```
Declare Function DelayTime Lib "PCI2006" (ByVal hDevice As Long, _
                                          ByVal nTime As Long) As Boolean
```

Delphi：

```
Function DelayTime (hDevice: Integer;
                    nTime : LongInt) : Boolean;
                    StdCall; External 'PCI2006' Name ' DelayTime';
```

LabVIEW：

请参考相关演示程序。

功能：微秒级延时函数。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

nTime 时间常数。单位 1 微秒。

返回值：若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获错误码。

相关函数：无。

第五节、文件对象操作函数原型说明

◆ 创建文件对象

函数原型：

Visual C++ & C++ Builder：

```
HANDLE CreateFileObject ( HANDLE hDevice,
                          LPCTSTR NewFileName,
                          int Mode)
```

Visual Basic：

```
Declare Function CreateFileObject Lib "PCI2006" (ByVal hDevice As Long, _
                                                 ByVal NewFileName As String, _
                                                 ByVal Mode As Integer) As Long
```

Delphi：

```
Function CreateFileObject (hDevice : Integer;
                           NewFileName : string;
                           Mode : Integer) : Integer;
                           Stdcall; external 'PCI2006' Name ' CreateFileObject ';
```

LabVIEW：

请参见相关演示程序。

功能：初始化设备文件对象，以期待 WriteFile 请求准备文件对象进行文件操作。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

NewFileName 新文件名。

Mode 文件操作方式，所用的文件操作方式控制字定义如下(可通过或指令实现多种方式并操作)：

常量名	常量值	功能定义
PCI2006_modeRead	0x0000	只读文件方式
PCI2006_modeWrite	0x0001	只写文件方式
PCI2006_modeReadWrite	0x0002	既读又写文件方式
PCI2006_modeCreate	0x1000	如果文件不存在可以创建该文件，　如果存在，则重建此文件，且清 0
PCI2006_typeText	0x4000	以文本方式操作文件

返回值：若成功，则返回文件对象句柄。

相关函数：　CreateDevice　　CreateFileObject　　WriteFile
　　　　　　ReadFile　　ReleaseFile　　ReleaseDevice

◆ 通过设备对象，往指定磁盘上写入用户空间的采样数据

函数原型：

Visual C++ & C++ Builder:

```
BOOL WriteFile(HANDLE hFileObject,
```

```
                    PVOID pDataBuffer,
                    ULONG nWriteSizeBytes)
```

Visual Basic:

```
Declare Function WriteFile Lib "PCI2006" ( ByVal hFileObject As Long,_
                                            ByRef pDataBuffer As Integer,_
                                            ByVal nWriteSizeBytes As Long) As Boolean
```

Delphi:

```
Function WriteFile(hFileObject: Integer;
                    pDataBuffer: Pointer;
                    nWriteSizeBytes : LongWord) : Boolean;
                    Stdcall; external 'PCI2006' Name ' WriteFile ';
```

LabVIEW:

详见相关演示程序。

功能：通过向设备对象发送“写磁盘消息”，设备对象便会以最快的速度完成写操作。注意为了保证写入的数据是可用的，这个操作将与用户程序保持同步，但与设备对象中的环形内存池操作保持异步，以得到更高的数据吞吐量，其文件名及路径应由CreateFileObject函数中的strFileName指定。

参数：

hFileObject 设备对象句柄，它应由CreateFileObject创建。

pDataBuffer 用户数据空间地址，可以是用户分配的数组空间。

nWriteSizeBytes 告诉设备对象往磁盘上一次写入数据的长度(以字节为单位)。

返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用GetLastErrorEx捕获错误码。

相关函数： CreateFileObject　WriteFile　ReadFile　ReleaseFile

- **通过设备对象,从指定磁盘文件中读采样数据**

函数原型：

Visual C++ & C++ Builder:

```
BOOL ReadFile( HANDLE hFileObject,
                PVOID pDataBuffer,
                ULONG OffsetBytes,
                ULONG nReadSizeBytes)
```

Visual Basic:

```
Declare Function ReadFile Lib "PCI2006" (  ByVal hFileObject As Long, _
                                            ByRef pDataBuffer As Integer, _
                                            ByVal OffsetBytes As Long, _
                                            ByVal nReadSizeBytes As Long) As Boolean
```

Delphi:

```
Function ReadFile(hFileObject : Integer;
                   pDataBuffer: Pointer;
                   OffsetBytes : LongWord;
                   nReadSizeBytes : LongWord) : Boolean;
                   Stdcall; external 'PCI2006' Name ' ReadFile ';
```

LabVIEW:

详见相关演示程序。

功能：将磁盘数据从指定文件中读入用户内存空间中，其访问方式可由用户在创建文件对象时指定。

参数：

hFileObject 设备对象句柄，它应由CreateFileObject创建。

pDataBuffer 用于接受文件数据的用户缓冲区指针，可以是用户分配的数组空间。

OffsetBytes 指定从文件开始端所偏移的读位置。

nReadSizeBytes 告诉设备对象从磁盘上一次读入数据的长度(以字为单位)。

返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用GetLastErrorEx捕获错误码。

相关函数： CreateFileObject　WriteFile　ReadFile　ReleaseFile

- **设置文件偏移位置**

函数原型：

Visual C++ & C++ Builder:

```
BOOL SetFileOffset (HANDLE hFileObject,
                    ULONG nOffsetBytes)
```

Visual Basic:

```
Declare Function SetFileOffset Lib "PCI2006" (ByVal hFileObject As Long,_
                                    ByVal nOffsetBytes As Long) As Boolean
```

Delphi:

```
Function SetFileOffset (hFileObject : Integer;
                        nOffsetBytes : LongWord):Boolean;
                        stdcall; external ' PCI2006 ' Name ' SetFileOffset ';
```

LabVIEW:

请参考相关演示程序。

功能：设置文件偏移位置，用它可以定位读写起点。

参数：hFileObject 文件对象句柄，它应由CreateFileObject创建。

返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用GetLastErrorEx捕获错误码。

相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

- ◆ 取得文件长度（字节）

函数原型：

Visual C++ & C++ Builder:

```
ULONG GetFileLength (HANDLE hFileObject)
```

Visual Basic:

```
Declare Function GetFileLength Lib "PCI2006" (ByVal hFileObject As Long) As Long
```

Delphi:

```
Function GetFileLength (hFileObject : Integer) : LongWord;
                        Stdcall; external 'PCI2006' Name ' GetFileLength ';
```

LabVIEW:

详见相关演示程序。

功能：取得文件长度。

参数：hFileObject 设备对象句柄，它应由CreateFileObject创建。

返回值：若成功，则返回>1，否则返回 0，用户可以用GetLastErrorEx捕获错误码。

相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

- ◆ 释放设备文件对象

函数原型：

Visual C++ & C++ Builder:

```
BOOL ReleaseFile(HANDLE hFileObject)
```

Visual Basic:

```
Declare Function ReleaseFile Lib "PCI2006" (ByVal hFileObject As Long) As Boolean
```

Delphi:

```
Function ReleaseFile(hFileObject : Integer) : Boolean;
                    Stdcall; external 'PCI2006' Name ' ReleaseFile ';
```

LabVIEW:

详见相关演示程序。

功能：释放设备文件对象。

参数：hFileObject 设备对象句柄，它应由CreateFileObject创建。

返回值：若成功，则返回TRUE，否则返回FALSE，用户可以用GetLastErrorEx捕获错误码。

相关函数： CreateFileObject WriteFile ReadFile ReleaseFile

- ◆ 取得指定磁盘的可用空间

函数原型：

Visual C++ & C++ Builder:

ULONGLONG GetDiskFreeBytes(LPCTSTR DiskName)

Visual Basic:

Declare Function GetDiskFreeBytes Lib "PCI2006" (ByVal DiskName As String) As Currency

Delphi:

Function GetDiskFreeBytes (DiskName: String) : Currency;
Stdcall; external 'PCI2006' Name ' GetDiskFreeBytes ';

LabVIEW:

GetDiskFreeBytes

DiskName abc U32 Return Disk Free Space

功能：取得指定磁盘的可用剩余空间(以字为单位)。

参数：DiskName 需要访问的盘符，若为 C 盘为"C:\\"，　D 盘为"D:\\"，以此类推。

返回值：若成功，返回大于或等于 0 的长整型值，否则返回零值，用户可用 GetLastErrorEx 捕获错误码。注意使用 64 位整型变量。